Common Language Infrastructure (CLI)

Partition VI:
Annexes

Annex A Introduction

Annex A this annex.

Annex B contains a number of sample programs written in CIL Assembly Language (ILAsm)

Annex C contains information about a particular implementation of an assembler, which provides a superset of the functionality of the syntax described in Partition II. It also provides a machine-readable description of the CIL instruction set which can be used to derive parts of the grammar used by this assembler as well as other tools that manipulate CIL.

Annex D contains a set of guidelines used in the design of the libraries of Partition IV. The rules are provided here since they have proven themselves effective in designing cross-language APIs. They also serve as guidelines for those intending to supply additional functionality in a way that meshes seamlessly with the standardized libraries.

Annex E contains information of interest to implementers with respect to the latitude they have in implementing the CLI.

Annex F contains information of interest to implementers with respect to relaxed fault handling.

Annex G shows several complete examples written using the parallel library.

Annex B Sample programs

This clause contains only informative text

This Annex shows several complete examples written using ILAsm.

B.1 Mutually recursive program (with tail calls)

The following is an example of a mutually recursive program that uses tail calls. The methods below determine whether a number is even or odd.

[*Example:*

```
.assembly extern mscorlib { }
.assembly test.exe { }
.class EvenOdd
{ .method private static bool IsEven(int32 N) cil managed
  { .maxstack   2
    ldarg.0             // N
    ldc.i4.0
    bne.un      NonZero
    ldc.i4.1
    ret

NonZero:
    ldarg.0
    ldc.i4.1
    sub
    tail.
    call bool EvenOdd::IsOdd(int32)
    ret
  } // end of method 'EvenOdd::IsEven'

  .method private static bool IsOdd(int32 N) cil managed
  { .maxstack   2
    // Demonstrates use of argument names and labels
    // Notice that the assembler does not convert these
    // automatically to their short versions
    ldarg       N
    ldc.i4.0
    bne.un      NonZero
    ldc.i4.0
    ret

NonZero:
    ldarg       N
    ldc.i4.1
    sub
    tail.

    call bool EvenOdd::IsEven(int32)
    ret
  } // end of method 'EvenOdd::IsOdd'

  .method public static void Test(int32 N) cil managed
  { .maxstack   1
    ldarg       N
    call        void [mscorlib]System.Console::Write(int32)
    ldstr       " is "
    call        void [mscorlib]System.Console::Write(string)
    ldarg       N
    call        bool EvenOdd::IsEven(int32)
    brfalse     LoadOdd
    ldstr       "even"

Print:
    call        void [mscorlib]System.Console::WriteLine(string)
    ret

LoadOdd:
    ldstr       "odd"
    br          Print
  } // end of method ‘EvenOdd::Test’
} // end of class ‘EvenOdd’

//Global method
.method public static void main() cil managed
{ .entrypoint
  .maxstack     1
  ldc.i4.5
  call          void EvenOdd::Test(int32)
  ldc.i4.2
  call          void EvenOdd::Test(int32)
  ldc.i4        100
  call          void EvenOdd::Test(int32)
  ldc.i4        1000001
  call          void EvenOdd::Test(int32)
  ret
} // end of global method ‘main’
```

end example]

B.2 Using value types

The following program shows how rational numbers can be implemented using value types.

[*Example:*

```
.assembly extern mscorlib { }
.assembly rational.exe { }
.class private sealed Rational extends [mscorlib]System.ValueType
         implements mscorlib]System.IComparable

{ .field public int32 Numerator
  .field public int32 Denominator

  .method virtual public int32 CompareTo(object o)
  // Implements IComparable::CompareTo(Object)
  { ldarg.0     // ‘this’ as a managed pointer
    ldfld int32 value class Rational::Numerator
    ldarg.1     // ‘o’ as an object
    unbox value class Rational
    ldfld int32 value class Rational::Numerator
    beq.s TryDenom
    ldc.i4.0
    ret

TryDenom:
    ldarg.0     // ‘this’ as a managed pointer
    ldfld int32 value class Rational::Denominator
    ldarg.1     // ‘o’ as an object
    unbox value class Rational
    ldfld int32 class Rational::Denominator
    ceq
    ret
  }
  .method virtual public string ToString()
  // Implements Object::ToString
  { .locals init (class [mscorlib]System.Text.StringBuilder SB,
                  string S, object N, object D)
    newobj void [mscorlib]System.Text.StringBuilder::.ctor()
    stloc.s SB
    ldstr "The value is: {0}/{1}"
    stloc.s S
    ldarg.0     // Managed pointer to self
    dup
    ldfld int32 value class Rational::Numerator
    box [mscorlib]System.Int32
    stloc.s N
    ldfld int32 value class Rational::Denominator
    box [mscorlib]System.Int32
    stloc.s D
    ldloc.s SB
    ldloc.s S
    ldloc.s N
    ldloc.s D
    call instance class [mscorlib]System.Text.StringBuilder
      [mscorlib]System.Text.StringBuilder::AppendFormat(string,
              object, object)
    callvirt instance string [mscorlib]System.Object::ToString()
    ret
  }
  .method public value class Rational Mul(value class Rational)
  {
    .locals init (value class Rational Result)
    ldloca.s Result
    dup
    ldarg.0     // ‘this’
    ldfld int32 value class Rational::Numerator
    ldarga.s    1     // arg
    ldfld int32 value class Rational::Numerator
    mul
    stfld int32 value class Rational::Numerator
    ldarg.0     // this
    ldfld int32 value class Rational::Denominator
    ldarga.s    1     // arg
    ldfld int32 value class Rational::Denominator
    mul
    stfld int32 value class Rational::Denominator
    ldloc.s Result
    ret
  }
}
.method static void main()
{
  .entrypoint
  .locals init (value class Rational Half,
                value class Rational Third,
                value class Rational Temporary,
                object H, object T)
  // Initialize Half, Third, H, and T
  ldloca.s Half
  dup
  ldc.i4.1
  stfld int32 value class Rational::Numerator
  ldc.i4.2
  stfld  int32 value class Rational::Denominator
  ldloca.s Third
  dup

  ldc.i4.1
  stfld int32 value class Rational::Numerator
  ldc.i4.3
  stfld int32 value class Rational::Denominator
  ldloc.s Half
  box value class Rational
  stloc.s H
  ldloc.s Third
  box value class Rational
  stloc.s T
  // WriteLine(H.IComparable::CompareTo(H))
  // Call CompareTo via interface using boxed instance

  ldloc H
  dup
  callvirt int32 [mscorlib]System.IComparable::CompareTo(object)
  call void [mscorlib]System.Console::WriteLine(bool)
  // WriteLine(Half.CompareTo(T))
  // Call CompareTo via value type directly
  ldloca.s Half
  ldloc T
  call instance int32
  value class Rational::CompareTo(object)
  call void [mscorlib]System.Console::WriteLine(bool)

  // WriteLine(Half.ToString())
  // Call virtual method via value type directly
  ldloca.s Half
  call instance string class Rational::ToString()
  call void [mscorlib]System.Console::WriteLine(string)

  // WriteLine(T.ToString)
  // Call virtual method inherited from Object, via boxed
instance
  ldloc T
  callvirt string [mscorlib]System.Object::ToString()
  call void [mscorlib]System.Console::WriteLine(string)
  // WriteLine((Half.Mul(T)).ToString())
  // Mul is called on two value types, returning a value type
  // ToString is then called directly on that value type

  // Note that we are required to introduce a temporary variable
  //   since the call to ToString requires
  //   a managed pointer (address)
  ldloca.s Half
  ldloc.s Third
  call instance value class Rational
         Rational::Mul(value class Rational)

  stloc.s Temporary
  ldloca.s Temporary
  call instance string Rational::ToString()
  call void [mscorlib]System.Console::WriteLine(string)
  ret
}
```

end example]

B.3 Custom attributes

[*Example:*

This subclause includes many example uses of custom attributes to help clarify the grammar and rules described above. The examples are written in C#, and each one shows a collection of one or more attributes, applied to a class (called “App”). The hex and ‘translation’ of the custom attribute blobs are shown as comments. The following abbreviations are used:

- FIELD = ELEMENT_TYPE_FIELD
- PROPERTY = 0x54
- STRING = ELEMENT_TYPE_STRING
- SZARRAY = ELEMENT_TYPE_SZARRAY
- U1 = ELEMENT_TYPE_U1
- I4 = ELEMENT_TYPE_I4
- OBJECT = 0x51

```
//
*****************************************************************
***************
// CustomSimple.cs
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class B : Attribute { public B(int i, ushort u) {} }

[B(7,9)]    // 01 00          // Prolog
            // 07 00 00 00    // 0x00000007
            // 09 00          // 0x0009
            // 00 00          // NumNamed
class App { static void Main() {} }

//
*****************************************************************
***************
// CustomString.cs
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class A : Attribute {
   public  string field;        // field
   private string back;         // backing field for property
   public  string prop {        // property
      get { return back;  }
      set { back = value; }
   }
   public  A(string x) {}       // ctor
}
[A(null)]   // 01 00            // Prolog
            // FF               // null
            // 00 00            // NumNamed

[A("")]     // 01 00            // Prolog
            // 00               // zero-length string
            // 00 00            // NumNamed

[A("ab",field="cd",prop="123")]  // 01 00             // Prolog
                                 // 02 61 62          // "ab"
                                 // 02 00             // NumNamed
                                 // 53 0e             // FIELD,
STRING
                                 // 05 66 69 65 6c 64 // "field"
as counted-UTF8
```

```
                                 // 02 63 64          // "cd" as
counted-UTF8
                                 // 54 0e             //
PROPERTY, STRING
                                 // 04 70 72 6f 70    // "prop"
as counted-UTF8
                                 // 03 31 32 33       // "123" as
counted-UTF8
class App { static void Main() {} }

//
*****************************************************************
***************
// CustomType.cs
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class C : Attribute {
   public C(Type t) {}
}
[C(typeof(C))]
// 01 00                                              // Prolog
// 01 43                                              // "C" as
counted-UTF8
// 00 00                                              // NumNamed

[C(typeof(string))]
// 01 00                                              // Prolog
// 0d 53 79 73 74 65 6d 2e 53 74 72 69 6e 67          //
"System.String" as counted-UTF8
// 00 00                                              // NumNamed

[C(typeof(System.Windows.Forms.Button))]
// 01 00                                              // Prolog
// 76 53 79 73 74 65 6d 2e 57 69 6e 64 6f 77          //
"System.Window
// 73 2e 46 6f 72 6d 73 2e 42 75 74 74 6f 6e 2c 53    //
s.Forms.Button,S
// 79 73 74 65 6d 2e 57 69 6e 64 6f 77 73 2e 46 6f    //
ystem.Windows.Fo
// 72 6d 73 2c 20 56 65 72 73 69 6f 6e 3d 32 2e 30    // rms,
Version=2.0
// 2e 33 36 30 30 2e 30 2c 20 43 75 6c 74 75 72 65    // .3600.0,
Culture
// 3d 6e 65 75 74 72 61 6c 2c 20 50 75 62 6c 69 63    //
=neutral, Public
// 4b 65 79 54 6f 6b 65 6e 3d 62 37 37 61 35 63 35    //
KeyToken=b77a5c5
// 36 31 39 33 34 65 30 38 39 00 00                   //
61934e089"
// 00 00                                              // NumNamed
class App { static void Main() {} }
```

Notice how various types are ‘stringified’: if the type is defined in the local assembly, or in mscorlib, then only its full name is required; if the type is defined in a different assembly, then its fully-qualified assembly name is required, includeing Version, Culture and PublicKeyToken, if non-defaulted.

```
//
*****************************************************************
***************
// CustomByteArray.cs
```

```
using System;
class D : Attribute {
   public  byte[] field;                              // field
   private byte[] back;                               // backing
field for property
   public  byte[] prop {                              // property
      get { return back;  }
      set { back = value; }
   }
   public D(params byte[] bs) {}                      // ctor
}
[D(1,2, field=new byte[]{3,4},prop=new byte[]{5})]
// 01 00                                              // Prolog
// 02 00 00 00                                        // NumElem
// 01 02                                              // 1,2
// 02 00                                              // NumNamed
// 53 1d 05                                           // FIELD,
SZARRAY, U1
// 05 66 69 65 6c 64                                  // "field"
as counted-UTF8
// 02 00 00 00                                        // NumElem =
0x00000002
// 03 04                                              // 3,4
// 54 1d 05                                           // PROPERTY,
SZARRAY, U1
// 04 70 72 6f 70                                     // "prop" as
counted-UTF8
// 01 00 00 00                                        // NumElem =
0x00000001
// 05                                                 // 5
class App { static void Main() {} }

//
******************************************************************
***************
// CustomBoxedValuetype.cs
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class E : Attribute {
   public object obj;                            // field called
"obj"
   public object o {                             // property
called "o"
      get { return o; }
      set { o = value; }
   }
   public E() {}                                 // default ctor
   public E(object x) {}
}

[E(42)]                                          // boxed 42
// 01 00                                         // Prolog
// 08                                            // I4
// 2a 00 00 00                                   // 0x0000002A
// 00 00                                         // NumNamed

[E(obj=7)]                                       // named field
// 01 00                                         // Prolog
// 01 00                                         // NumNamed
// 53 51                                         // FIELD, OBJECT
// 03 6f 62 6a                                     // "obj" as
counted-UTF8
// 08                                              // I4
// 07 00 00 00                                     // 0x00000007

[E(o=0xEE)]                                        // named property
// 01 00                                           // Prolog
// 01 00                                           // NumNamed
// 54 51                                           // PROPERTY,
OBJECT
// 01 6f                                           // "o" as
counted-UTF8
// 08                                              // I4
// ee 00 00 00                                     // 0x000000EE
class App { static void Main() {} }
```

This example illustrates how to construct blobs for a custom attribute that accepts a `System.Object` in its constructor, as a named field, and as a named property. In each case, the argument given is an `int32`, which is boxed automatically by the C# compiler.

Notice the OBJECT = 0x51 byte. This is emitted for “named” fields or properties of type `System.Object`.

```
//
*****************************************************************
***************
// CustomShortArray.cs
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class F : Attribute {
   public F(params short[] cs) {}    // ctor
}
//[F()]
// 01 00                             // Prolog
// 00 00 00 00                       // NumElem
// 00 00                             // NumNamed

//[F(null)]
// 01 00                             // Prolog
// ff ff ff ff                       // NumElem = -1 => null
// 00 00                             // NumNamed

[F(1,2)]
// 01 00                             // Prolog
// 02 00 00 00                       // NumElem
// 01 00 02 00                       // 0x0001, 0x0002
// 00 00                             // NumNamed
class App { static void Main() {} }
```

end example]

B.4 Generics code and metadata

The following informative text, shows a partial implementation for a naive phone-book class. It shows the source first, as written in ILAsm, followed by the equivalent (much shorter) code, written in C#. The section then goes on to examine the metadata generated for this code.

B.4.1 ILAsm version

```
.assembly extern mscorlib {}
.assembly Phone {}
```

```
.class private Phone`2<([mscorlib]System.Object) K,
([mscorlib]System.Object) V>
    extends  [mscorlib]System.Object {
   .field private int32 hi
   .field private !0[]  keys
   .field private !1[]  vals
   .method public instance void Add(!0 k, !1 v) {
      .maxstack  4
      .locals init (int32 temp)
      ldarg.0
      ldfld      !0[] class Phone`2<!0,!1>::keys
      ldarg.0
      dup
      ldfld      int32 class Phone`2<!0,!1>::hi
      ldc.i4.1
      add
      dup
      stloc.0
      stfld      int32 class Phone`2<!0,!1>::hi
      ldloc.0
      ldarg.1
      stelem     !0
      ldarg.0
      ldfld      !1[] class Phone`2<!0,!1>::vals
      ldarg.0
      ldfld      int32 class Phone`2<!0,!1>::hi
      ldarg.2
      stelem     !1
      ret
   }  // end of Method Add
}  // end of class Phone

.class App extends [mscorlib]System.Object {
   .method static void Main() {
      .entrypoint
      .maxstack  3
      .locals init (class Phone`2<string,int32> temp)
      newobj     instance void class
        Phone`2<string,int32>::.ctor()
      stloc.0
      ldloc.0
      ldstr      "Jim"
      ldc.i4.7
      callvirt   instance void class
        Phone`2<string,int32>::Add(!0, !1)
      ret
   }  // end of method Main
}  // end of class App
```

B.4.2 C# version

```
using System;

class Phone<K,V> {
   private int hi = -1;
   private K[] keys;
   private V[] vals;
   public Phone() { keys = new K[10]; vals = new V[10]; }
   public void Add(K k, V v) { keys[++hi] = k; vals[hi] = v; }
}

class App {
   static void AddOne<KK,VV>(Phone<KK,VV> phone, KK kk, VV vv) {
      phone.Add(kk, vv);
   }
   static void Main() {
      Phone<string, int> d = new Phone<string, int>();
      d.Add("Jim", 7);
      AddOne(d, "Joe", 8);
   }
}
```

B.4.3 Metadata

As detailed in §23.2.12 of Partition II, the *Type* non-terminal now includes a production for generic instantiations, as follows:

```
Type ::= . . .
     | GENERICINST (CLASS | VALUETYPE) TypeDefOrRefEncoded GenArgCount Type *
```

Following this production, the `Phone<string,int>` instantiation above is encoded as:

```
0x15  ELEMENT_TYPE_GENERICINST
0x12  ELEMENT_TYPE_CLASS
0x08  TypeDefOrRef coded index for class "Phone<K,V>"
0x02  GenArgCount = 2
0x0E     ELEMENT_TYPE_STRING
0x08     ELEMENT_TYPE_I4
```

Similarly, the signature for the field `vals` is encoded as:

```
0x06  FIELD
0x1D  ELEMENT_TYPE_SZARRAY
0x13  ELEMENT_TYPE_VAR
0x01  1, representing generic argument number 1 (i.e., "V")
```

Similarly, the signature for the (rather contrived) static method `AddOne` is encoded as:

```
0x10  IMAGE_CEE_CS_CALLCONV_GENERIC
0x02  GenParamCount = 2 (2 generic parameters for this method:
KK and VV
0x03  ParamCount = 3 (phone, kk and vv)
0x01  RetType = ELEMENT_TYPE_VOID
0x15  Param-0:  ELEMENT_TYPE_GENERICINST
0x12            ELEMENT_TYPE_CLASS
0x08            TypeDefOrRef coded index for class
"Phone<KK,VV>"
0x02            GenArgCount = 2
0x1e               ELEMENT_TYPE_MVAR
0x00               !!0 (KK in AddOne<KK,VV>)
0x1e               ELEMENT_TYPE_MVAR
0x01               !!1 (VV in AddOne<KK,VV>)
0x1e  Param-1   ELEMENT_TYPE_MVAR
0x00            !!0 (KK in AddOne<KK,VV>)
0x1e  Param-2   ELEMENT_TYPE_MVAR
0x01            !!1 (VV in AddOne<KK,VV>)
```

Notice that the above example uses indenting to help denote loops over the three method parameters, and the two generic parameters on `Phone`.

Annex C CIL assembler implementation

This clause contains only informative text

This clause provides information about a particular assembler for CIL, called *ilasm*. It supports a superset of the syntax defined normatively in Partition II, and provides a concrete syntax for the CIL instructions specified in Partition III.

Even for those who have no interest in this particular assembler, §C.1 and §C.3 might be of interest. The former is a machine-readable file (ready for input to a C or C++ preprocessor) that partially describes the CIL instructions. It can be used to generate tables for use by a wide variety of tools that deal with CIL. The latter contains a concrete syntax for CIL instructions, which is not described elsewhere.

C.1 ILAsm keywords

This subclause provides a complete list of the keywords used by *ilasm*. If users wish to use any of these as simple identifiers within programs they just make use of the appropriate escape notation (single or double quotation marks as specified in the grammar). This assembler is case-sensitive.

#line
.addon
.assembly
.cctor
.class
.corflags
.ctor
.custom
.data
.emitbyte
.entrypoint
.event
.export
.field
.file
.fire
.get
.hash
.imagebase
.import
.language
.line
.locale
.localized
.locals
.manifestres
.maxstack
.method
.module
.mresource
.namespace
.other
.override
.pack
.param
.pdirect
.permission
.permissionset
.property
.publickey
.publickeytoken
.removeon
.set
.size
.subsystem
.try
.ver
.vtable
.vtentry
.vtfixup
.zeroinit
^THE_END^
abstract
add
add.ovf
add.ovf.un
algorithm
alignment
and
ansi
any
arglist
array
as
assembly
assert
at
auto
autochar
beforefieldinit
beq
beq.s
bge
bge.s
bge.un
bge.un.s
bgt
bgt.s
bgt.un
bgt.un.s
ble
ble.s
ble.un
ble.un.s
blob
blob_object
blt
blt.s
blt.un
blt.un.s
bne.un
bne.un.s
bool
box
br
br.s

break
brfalse
brfalse.s
brinst
brinst.s
brnull
brnull.s
brtrue
brtrue.s
brzero
brzero.s
bstr
bytearray
byvalstr
call
calli
callmostderived
callvirt
carray
castclass
catch
cdecl
ceq
cf
cgt
cgt.un
char
cil
ckfinite
class
clsid
clt
clt.un
const
constrained.
conv.i
conv.i1
conv.i2

conv.i4
conv.i8
conv.ovf.i
conv.ovf.i.un
conv.ovf.i1
conv.ovf.i1.un
conv.ovf.i2
conv.ovf.i2.un
conv.ovf.i4
conv.ovf.i4.un
conv.ovf.i8
conv.ovf.i8.un
conv.ovf.u
conv.ovf.u.un
conv.ovf.u1
conv.ovf.u1.un
conv.ovf.u2
conv.ovf.u2.un
conv.ovf.u4
conv.ovf.u4.un
conv.ovf.u8
conv.ovf.u8.un
conv.r.un
conv.r4
conv.r8
conv.u
conv.u1
conv.u2
conv.u4
conv.u8
cpblk
cpobj
currency
custom
date
decimal
default
default

demand
deny
div
div.un
dup
endfault
endfilter
endfinally
endmac
enum
error
explicit
extends
extern
false
famandassem
family
famorassem
fastcall
fastcall
fault
field
filetime
filter
final
finally
fixed
float
float32
float64
forwardref
fromunmanaged
handler
hidebysig
hresult
idispatch
il
illegal

implements
implicitcom
implicitres
import
in
inheritcheck
init
initblk
initobj
initonly
instance
int
int16
int32
int64
int8
interface
internalcall
isinst
iunknown
jmp
lasterr
lcid
ldarg
ldarg.0
ldarg.1
ldarg.2
ldarg.3
ldarg.s
ldarga
ldarga.s
ldc.i4
ldc.i4.0
ldc.i4.1
ldc.i4.2
ldc.i4.3
ldc.i4.4
ldc.i4.5

ldc.i4.6
ldc.i4.7
ldc.i4.8
ldc.i4.M1
ldc.i4.m1
ldc.i4.s
ldc.i8
ldc.r4
ldc.r8
ldelem
ldelem.i
ldelem.i1
ldelem.i2
ldelem.i4
ldelem.i8
ldelem.r4
ldelem.r8
ldelem.ref
ldelem.u1
ldelem.u2
ldelem.u4
ldelem.u8
ldelema
ldfld
ldflda
ldftn
ldind.i
ldind.i1
ldind.i2
ldind.i4
ldind.i8
ldind.r4
ldind.r8
ldind.ref
ldind.u1
ldind.u2
ldind.u4
ldind.u8
ldlen
ldloc
ldloc.0
ldloc.1
ldloc.2
ldloc.3
ldloc.s
ldloca
ldloca.s
ldnull
ldobj
ldsfld
ldsflda
ldstr
ldtoken
ldvirtftn
leave
leave.s
linkcheck
literal
localloc
lpstr
lpstruct
lptstr
lpvoid
lpwstr
managed
marshal
method
mkrefany
modopt
modreq
mul
mul.ovf
mul.ovf.un
native
neg
nested
newarr
newobj
newslot
noappdomain
no.
noinlining
nomachine
nomangle
nometadata
noncasdemand
noncasinheritance
noncaslinkdemand
nop
noprocess
not
not_in_gc_heap
notremotable
notserialized
null
nullref
object
objectref
opt
optil
or
out
permitonly
pinned
pinvokeimpl
pop
prefix1
prefix2
prefix3
prefix4
prefix5
prefix6
prefix7
prefixref
prejitdeny
prejitgrant
preservesig
private
privatescope
protected
public
readonly.
record
refany
refanytype
refanyval
rem
rem.un
reqmin
reqopt
reqrefuse
reqsecobj
request
ret
rethrow
retval
rtspecialname
runtime
safearray
sealed
sequential
serializable
shl
shr
shr.un
sizeof
special
specialname
starg
starg.s
static
stdcall

stdcall
stelem
stelem.i
stelem.i1
stelem.i2
stelem.i4
stelem.i8
stelem.r4
stelem.r8
stelem.ref
stfld
stind.i
stind.i1
stind.i2
stind.i4
stind.i8
stind.r4
stind.r8
stind.ref
stloc
stloc.0
stloc.1
stloc.2
stloc.3
stloc.s
stobj
storage
stored_object
stream
streamed_object
string
struct
stsfld
sub
sub.ovf
sub.ovf.un
switch
synchronized
syschar
sysstring
tail.
tbstr
thiscall
thiscall
throw
tls
to
true
typedref
unaligned.
unbox
unbox.any
unicode
unmanaged
unmanagedexp
unsigned
unused
userdefined
value
valuetype
vararg
variant
vector
virtual
void
volatile.
wchar
winapi
with
wrapper
xor

C.2 CIL opcode descriptions

This subclause contains text, which is intended for use with the C or C++ preprocessor. By appropriately defining the macros `OPDEF` and `OPALIAS` before including this text, it is possible to use this to produce tables or code for handling CIL instructions.

The `OPDEF` macro is passed 10 arguments, in the following order:

1. A symbolic name for the opcode, beginning with CEE_
2. A string that constitutes the name of the opcode and corresponds to the names given in Partition III.
3. Data removed from the stack to compute this operations result. The possible values here are the following:
    a. Pop0 – no inputs
    b. Pop1 – one value type specified by data flow
    c. Pop1+Pop1 – two input values, types specified by data flow
    d. PopI – one machine-sized integer
    e. PopI+Pop1 – Top of stack is described by data flow, next item is a native pointer
    f. PopI+PopI – Top two items on stack are integers (size can vary by instruction)
    g. PopI+PopI+PopI – Top three items on stack are machine-sized integers
    h. PopI8+Pop8 – Top of stack is an 8-byte integer, next is a native pointer
    i. PopI+PopR4 – Top of stack is a 4-byte floating point number, next is a native pointer

    j. PopI+PopR8 – Top of stack is an 8-byte floating point number, next is a native pointer
    k. PopRef – Top of stack is an object reference
    l. PopRef+PopI – Top of stack is an integer (size can vary by instruction), next is an object reference
    m. PopRef+PopI+PopI – Top of stack has two integers (size can vary by instruction), next is an object reference
    n. PopRef+PopI+PopI8 – Top of stack is an 8-byte integer, then a native-sized integer, then an object reference
    o. PopRef+PopI+PopR4 – Top of stack is an 4-byte floating point number, then a native-sized integer, then an object reference
    p. PopRef+PopI+PopR8 – Top of stack is an 8-byte floating point number, then a native-sized integer, then an object reference
    q. VarPop – variable number of items used, see Partition III for details
4. Amount and type of data pushed as a result of the instruction. The possible values here are the following:
    a. Push0 – no output value
    b. Push1 – one output value, type defined by data flow.
    c. Push1+Push1 – two output values, type defined by data flow
    d. PushI – push one native integer or pointer
    e. PushI8 – push one 8-byte integer
    f. PushR4 – push one 4-byte floating point number
    g. PushR8 – push one 8-byte floating point number
    h. PushRef – push one object reference
    i. VarPush – variable number of items pushed, see Partition III for details
5. Type of in-line argument to instruction. The in-line argument is stored with least significant byte first (“little endian”). The possible values here are the following:
    a. InlineBrTarget – Branch target, represented as a 4-byte signed integer from the beginning of the instruction following the current instruction.
    b. InlineField – Metadata token (4 bytes) representing a FieldRef (i.e., a MemberRef to a field) or FieldDef
    c. InlineI – 4-byte integer
    d. InlineI8 – 8-byte integer
    e. InlineMethod – Metadata token (4 bytes) representing a MethodRef (i.e., a MemberRef to a method) or MethodDef
    f. InlineNone – No in-line argument
    g. InlineR – 8-byte floating point number
    h. InlineSig – Metadata token (4 bytes) representing a standalone signature
    i. InlineString – Metadata token (4 bytes) representing a UserString
    j. InlineSwitch – Special for the switch instructions, see Partition III for details
    k. InlineTok – Arbitrary metadata token (4 bytes) , used for ldtoken instruction, see Partition III for details
    l. InlineType – Metadata token (4 bytes) representing a TypeDef, TypeRef, or TypeSpec

    m. InlineVar – 2-byte integer representing an argument or local variable
    n. ShortInlineBrTarget – Short branch target, represented as 1 signed byte from the beginning of the instruction following the current instruction.
    o. ShortInlineI – 1-byte integer, signed or unsigned depending on instruction
    p. ShortInlineR – 4-byte floating point number
    q. ShortInlineVar – 1-byte integer representing an argument or local variable

6. Type of opcode. The current classification is of no current value, but is retained for historical reasons.
7. Number of bytes for the opcode. Currently 1 or 2, can be 4 in future
8. First byte of 2-byte encoding, or 0xFF if single byte instruction.
9. One byte encoding, or second byte of 2-byte encoding.
10. Control flow implications of instruction. The possible values here are the following:
    a. BRANCH – unconditional branch
    b. CALL – method call
    c. COND_BRANCH – conditional branch
    d. META – unused operation or prefix code
    e. NEXT – control flow unaltered (“fall through”)
    f. RETURN – return from method
    g. THROW – throw or rethrow an exception

The `OPALIAS` macro takes three arguments:

1. A symbolic name for a “new instruction” which is simply an alias (renaming for the assembler) of an existing instruction.
2. A string name for the “new instruction.”
3. The symbolic name for an instruction introduced using the `OPDEF` macro. The “new instruction” is really just an alternative name for this instruction.

```
#ifndef __OPCODE_DEF_
#define __OPCODE_DEF_

#define MOOT    0x00    // Marks unused second byte when encoding single
#define STP1    0xFE    // Prefix code 1 for Standard Map
#define REFPRE  0xFF    // Prefix for Reference Code Encoding
#define RESERVED_PREFIX_START 0xF7

#endif

// If the first byte of the standard encoding is 0xFF, then
// the second byte can be used as 1 byte encoding.  Otherwise
l   b         b
// the encoding is two bytes.
e   y         y
//
n   t         t
//
g   e         e
//
(unused)       t
//  Canonical Name                    String Name              Stack Behaviour
Operand Params    Opcode Kind      h   1         2    Control Flow
// ---------------------------------------------------------------------------------
--------------------------------------------------------------------
OPDEF(CEE_NOP,                        "nop",              Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x00,    NEXT)
OPDEF(CEE_BREAK,                      "break",            Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x01,    BREAK)
```

```
OPDEF(CEE_LDARG_0,                    "ldarg.0",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x02,    NEXT)
OPDEF(CEE_LDARG_1,                    "ldarg.1",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x03,    NEXT)
OPDEF(CEE_LDARG_2,                    "ldarg.2",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x04,    NEXT)
OPDEF(CEE_LDARG_3,                    "ldarg.3",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x05,    NEXT)
OPDEF(CEE_LDLOC_0,                    "ldloc.0",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x06,    NEXT)
OPDEF(CEE_LDLOC_1,                    "ldloc.1",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x07,    NEXT)
OPDEF(CEE_LDLOC_2,                    "ldloc.2",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x08,    NEXT)
OPDEF(CEE_LDLOC_3,                    "ldloc.3",          Pop0,               Push1,
InlineNone,         IMacro,      1,  0xFF,    0x09,    NEXT)
OPDEF(CEE_STLOC_0,                    "stloc.0",          Pop1,               Push0,
InlineNone,         IMacro,      1,  0xFF,    0x0A,    NEXT)
OPDEF(CEE_STLOC_1,                    "stloc.1",          Pop1,               Push0,
InlineNone,         IMacro,      1,  0xFF,    0x0B,    NEXT)
OPDEF(CEE_STLOC_2,                    "stloc.2",          Pop1,               Push0,
InlineNone,         IMacro,      1,  0xFF,    0x0C,    NEXT)
OPDEF(CEE_STLOC_3,                    "stloc.3",          Pop1,               Push0,
InlineNone,         IMacro,      1,  0xFF,    0x0D,    NEXT)
OPDEF(CEE_LDARG_S,                    "ldarg.s",          Pop0,               Push1,
ShortInlineVar,     IMacro,      1,  0xFF,    0x0E,    NEXT)
OPDEF(CEE_LDARGA_S,                   "ldarga.s",         Pop0,               PushI,
ShortInlineVar,     IMacro,      1,  0xFF,    0x0F,    NEXT)
OPDEF(CEE_STARG_S,                    "starg.s",          Pop1,               Push0,
ShortInlineVar,     IMacro,      1,  0xFF,    0x10,    NEXT)
OPDEF(CEE_LDLOC_S,                    "ldloc.s",          Pop0,               Push1,
ShortInlineVar,     IMacro,      1,  0xFF,    0x11,    NEXT)
OPDEF(CEE_LDLOCA_S,                   "ldloca.s",         Pop0,               PushI,
ShortInlineVar,     IMacro,      1,  0xFF,    0x12,    NEXT)
OPDEF(CEE_STLOC_S,                    "stloc.s",          Pop1,               Push0,
ShortInlineVar,     IMacro,      1,  0xFF,    0x13,    NEXT)
OPDEF(CEE_LDNULL,                     "ldnull",           Pop0,               PushRef,
InlineNone,         IPrimitive,  1,  0xFF,    0x14,    NEXT)
OPDEF(CEE_LDC_I4_M1,                  "ldc.i4.m1",        Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x15,    NEXT)
OPDEF(CEE_LDC_I4_0,                   "ldc.i4.0",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x16,    NEXT)
OPDEF(CEE_LDC_I4_1,                   "ldc.i4.1",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x17,    NEXT)
OPDEF(CEE_LDC_I4_2,                   "ldc.i4.2",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x18,    NEXT)
OPDEF(CEE_LDC_I4_3,                   "ldc.i4.3",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x19,    NEXT)
OPDEF(CEE_LDC_I4_4,                   "ldc.i4.4",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x1A,    NEXT)
OPDEF(CEE_LDC_I4_5,                   "ldc.i4.5",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x1B,    NEXT)
OPDEF(CEE_LDC_I4_6,                   "ldc.i4.6",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x1C,    NEXT)
OPDEF(CEE_LDC_I4_7,                   "ldc.i4.7",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x1D,    NEXT)
OPDEF(CEE_LDC_I4_8,                   "ldc.i4.8",         Pop0,               PushI,
InlineNone,         IMacro,      1,  0xFF,    0x1E,    NEXT)
OPDEF(CEE_LDC_I4_S,                   "ldc.i4.s",         Pop0,               PushI,
ShortInlineI,       IMacro,      1,  0xFF,    0x1F,    NEXT)
OPDEF(CEE_LDC_I4,                     "ldc.i4",           Pop0,               PushI,
InlineI,            IPrimitive,  1,  0xFF,    0x20,    NEXT)
OPDEF(CEE_LDC_I8,                     "ldc.i8",           Pop0,               PushI8,
InlineI8,           IPrimitive,  1,  0xFF,    0x21,    NEXT)
OPDEF(CEE_LDC_R4,                     "ldc.r4",           Pop0,               PushR4,
ShortInlineR,       IPrimitive,  1,  0xFF,    0x22,    NEXT)
OPDEF(CEE_LDC_R8,                     "ldc.r8",           Pop0,               PushR8,
InlineR,            IPrimitive,  1,  0xFF,    0x23,    NEXT)
OPDEF(CEE_UNUSED49,                   "unused",                      Pop0,
Push0,       InlineNone,         IPrimitive,  1,  0xFF,    0x24,    NEXT)
OPDEF(CEE_DUP,                        "dup",              Pop1,
Push1+Push1, InlineNone,         IPrimitive,  1,  0xFF,    0x25,    NEXT)
OPDEF(CEE_POP,                        "pop",              Pop1,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x26,    NEXT)
OPDEF(CEE_JMP,                        "jmp",              Pop0,               Push0,
InlineMethod,       IPrimitive,  1,  0xFF,    0x27,    CALL)
```

```
OPDEF(CEE_CALL,                       "call",             VarPop,             VarPush,
InlineMethod,       IPrimitive,  1,  0xFF,    0x28,    CALL)
OPDEF(CEE_CALLI,                      "calli",            VarPop,             VarPush,
InlineSig,          IPrimitive,  1,  0xFF,    0x29,    CALL)
OPDEF(CEE_RET,                        "ret",              VarPop,             Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x2A,    RETURN)
OPDEF(CEE_BR_S,                       "br.s",             Pop0,               Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x2B,    BRANCH)
OPDEF(CEE_BRFALSE_S,                  "brfalse.s",        PopI,               Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x2C,    COND_BRANCH)
OPDEF(CEE_BRTRUE_S,                   "brtrue.s",         PopI,               Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x2D,    COND_BRANCH)
OPDEF(CEE_BEQ_S,                      "beq.s",            Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x2E,    COND_BRANCH)
OPDEF(CEE_BGE_S,                      "bge.s",            Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x2F,    COND_BRANCH)
OPDEF(CEE_BGT_S,                      "bgt.s",            Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x30,    COND_BRANCH)
OPDEF(CEE_BLE_S,                      "ble.s",            Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x31,    COND_BRANCH)
OPDEF(CEE_BLT_S,                      "blt.s",            Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x32,    COND_BRANCH)
OPDEF(CEE_BNE_UN_S,                   "bne.un.s",         Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x33,    COND_BRANCH)
OPDEF(CEE_BGE_UN_S,                   "bge.un.s",         Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x34,    COND_BRANCH)
OPDEF(CEE_BGT_UN_S,                   "bgt.un.s",         Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x35,    COND_BRANCH)
OPDEF(CEE_BLE_UN_S,                   "ble.un.s",         Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x36,    COND_BRANCH)
OPDEF(CEE_BLT_UN_S,                   "blt.un.s",         Pop1+Pop1,          Push0,
ShortInlineBrTarget,IMacro,      1,  0xFF,    0x37,    COND_BRANCH)
OPDEF(CEE_BR,                         "br",               Pop0,               Push0,
InlineBrTarget,     IPrimitive,  1,  0xFF,    0x38,    BRANCH)
OPDEF(CEE_BRFALSE,                    "brfalse",          PopI,               Push0,
InlineBrTarget,     IPrimitive,  1,  0xFF,    0x39,    COND_BRANCH)
OPDEF(CEE_BRTRUE,                     "brtrue",           PopI,               Push0,
InlineBrTarget,     IPrimitive,  1,  0xFF,    0x3A,    COND_BRANCH)
OPDEF(CEE_BEQ,                        "beq",              Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x3B,    COND_BRANCH)
OPDEF(CEE_BGE,                        "bge",              Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x3C,    COND_BRANCH)
OPDEF(CEE_BGT,                        "bgt",              Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x3D,    COND_BRANCH)
OPDEF(CEE_BLE,                        "ble",              Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x3E,    COND_BRANCH)
OPDEF(CEE_BLT,                        "blt",              Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x3F,    COND_BRANCH)
OPDEF(CEE_BNE_UN,                     "bne.un",           Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x40,    COND_BRANCH)
OPDEF(CEE_BGE_UN,                     "bge.un",           Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x41,    COND_BRANCH)
OPDEF(CEE_BGT_UN,                     "bgt.un",           Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x42,    COND_BRANCH)
OPDEF(CEE_BLE_UN,                     "ble.un",           Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x43,    COND_BRANCH)
OPDEF(CEE_BLT_UN,                     "blt.un",           Pop1+Pop1,          Push0,
InlineBrTarget,     IMacro,      1,  0xFF,    0x44,    COND_BRANCH)
OPDEF(CEE_SWITCH,                     "switch",           PopI,               Push0,
InlineSwitch,       IPrimitive,  1,  0xFF,    0x45,    COND_BRANCH)
OPDEF(CEE_LDIND_I1,                   "ldind.i1",         PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x46,    NEXT)
OPDEF(CEE_LDIND_U1,                   "ldind.u1",         PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x47,    NEXT)
OPDEF(CEE_LDIND_I2,                   "ldind.i2",         PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x48,    NEXT)
OPDEF(CEE_LDIND_U2,                   "ldind.u2",         PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x49,    NEXT)
OPDEF(CEE_LDIND_I4,                   "ldind.i4",         PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x4A,    NEXT)
OPDEF(CEE_LDIND_U4,                   "ldind.u4",         PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x4B,    NEXT)
OPDEF(CEE_LDIND_I8,                   "ldind.i8",         PopI,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0x4C,    NEXT)
OPDEF(CEE_LDIND_I,                    "ldind.i",          PopI,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x4D,    NEXT)
```

```
OPDEF(CEE_LDIND_R4,                  "ldind.r4",         PopI,               PushR4,
InlineNone,         IPrimitive,  1,  0xFF,    0x4E,    NEXT)
OPDEF(CEE_LDIND_R8,                  "ldind.r8",         PopI,               PushR8,
InlineNone,         IPrimitive,  1,  0xFF,    0x4F,    NEXT)
OPDEF(CEE_LDIND_REF,                 "ldind.ref",        PopI,               PushRef,
InlineNone,         IPrimitive,  1,  0xFF,    0x50,    NEXT)
OPDEF(CEE_STIND_REF,                 "stind.ref",        PopI+PopI,          Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x51,    NEXT)
OPDEF(CEE_STIND_I1,                  "stind.i1",         PopI+PopI,          Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x52,    NEXT)
OPDEF(CEE_STIND_I2,                  "stind.i2",         PopI+PopI,          Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x53,    NEXT)
OPDEF(CEE_STIND_I4,                  "stind.i4",         PopI+PopI,          Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x54,    NEXT)
OPDEF(CEE_STIND_I8,                  "stind.i8",         PopI+PopI8,         Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x55,    NEXT)
OPDEF(CEE_STIND_R4,                  "stind.r4",         PopI+PopR4,         Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x56,    NEXT)
OPDEF(CEE_STIND_R8,                  "stind.r8",         PopI+PopR8,         Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x57,    NEXT)
OPDEF(CEE_ADD,                       "add",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x58,    NEXT)
OPDEF(CEE_SUB,                       "sub",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x59,    NEXT)
OPDEF(CEE_MUL,                       "mul",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x5A,    NEXT)
OPDEF(CEE_DIV,                       "div",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x5B,    NEXT)
OPDEF(CEE_DIV_UN,                    "div.un",           Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x5C,    NEXT)
OPDEF(CEE_REM,                       "rem",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x5D,    NEXT)
OPDEF(CEE_REM_UN,                    "rem.un",           Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x5E,    NEXT)
OPDEF(CEE_AND,                       "and",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x5F,    NEXT)
OPDEF(CEE_OR,                        "or",               Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x60,    NEXT)
OPDEF(CEE_XOR,                       "xor",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x61,    NEXT)
OPDEF(CEE_SHL,                       "shl",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x62,    NEXT)
OPDEF(CEE_SHR,                       "shr",              Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x63,    NEXT)
OPDEF(CEE_SHR_UN,                    "shr.un",           Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x64,    NEXT)
OPDEF(CEE_NEG,                       "neg",              Pop1,               Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x65,    NEXT)
OPDEF(CEE_NOT,                       "not",              Pop1,               Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0x66,    NEXT)
OPDEF(CEE_CONV_I1,                   "conv.i1",          Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x67,    NEXT)
OPDEF(CEE_CONV_I2,                   "conv.i2",          Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x68,    NEXT)
OPDEF(CEE_CONV_I4,                   "conv.i4",          Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x69,    NEXT)
OPDEF(CEE_CONV_I8,                   "conv.i8",          Pop1,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0x6A,    NEXT)
OPDEF(CEE_CONV_R4,                   "conv.r4",          Pop1,               PushR4,
InlineNone,         IPrimitive,  1,  0xFF,    0x6B,    NEXT)
OPDEF(CEE_CONV_R8,                   "conv.r8",          Pop1,               PushR8,
InlineNone,         IPrimitive,  1,  0xFF,    0x6C,    NEXT)
OPDEF(CEE_CONV_U4,                   "conv.u4",          Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x6D,    NEXT)
OPDEF(CEE_CONV_U8,                   "conv.u8",          Pop1,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0x6E,    NEXT)
OPDEF(CEE_CALLVIRT,                  "callvirt",         VarPop,             VarPush,
InlineMethod,       IObjModel,   1,  0xFF,    0x6F,    CALL)
OPDEF(CEE_CPOBJ,                     "cpobj",            PopI+PopI,          Push0,
InlineType,         IObjModel,   1,  0xFF,    0x70,    NEXT)
OPDEF(CEE_LDOBJ,                     "ldobj",            PopI,               Push1,
InlineType,         IObjModel,   1,  0xFF,    0x71,    NEXT)
OPDEF(CEE_LDSTR,                     "ldstr",            Pop0,               PushRef,
InlineString,       IObjModel,   1,  0xFF,    0x72,    NEXT)
OPDEF(CEE_NEWOBJ,                    "newobj",           VarPop,             PushRef,
InlineMethod,       IObjModel,   1,  0xFF,    0x73,    CALL)
```

```
OPDEF(CEE_CASTCLASS,                  "castclass",        PopRef,             PushRef,
InlineType,         IObjModel,   1,  0xFF,    0x74,    NEXT)
OPDEF(CEE_ISINST,                     "isinst",           PopRef,             PushI,
InlineType,         IObjModel,   1,  0xFF,    0x75,    NEXT)
OPDEF(CEE_CONV_R_UN,                  "conv.r.un",        Pop1,               PushR8,
InlineNone,         IPrimitive,  1,  0xFF,    0x76,    NEXT)
OPDEF(CEE_UNUSED58,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x77,    NEXT)
OPDEF(CEE_UNUSED1,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0x78,    NEXT)
OPDEF(CEE_UNBOX,                      "unbox",            PopRef,             PushI,
InlineType,         IPrimitive,  1,  0xFF,    0x79,    NEXT)
OPDEF(CEE_THROW,                      "throw",            PopRef,             Push0,
InlineNone,         IObjModel,   1,  0xFF,    0x7A,    THROW)
OPDEF(CEE_LDFLD,                      "ldfld",            PopRef,             Push1,
InlineField,        IObjModel,   1,  0xFF,    0x7B,    NEXT)
OPDEF(CEE_LDFLDA,                     "ldflda",           PopRef,             PushI,
InlineField,        IObjModel,   1,  0xFF,    0x7C,    NEXT)
OPDEF(CEE_STFLD,                      "stfld",            PopRef+Pop1,        Push0,
InlineField,        IObjModel,   1,  0xFF,    0x7D,    NEXT)
OPDEF(CEE_LDSFLD,                     "ldsfld",           Pop0,               Push1,
InlineField,        IObjModel,   1,  0xFF,    0x7E,    NEXT)
OPDEF(CEE_LDSFLDA,                    "ldsflda",          Pop0,               PushI,
InlineField,        IObjModel,   1,  0xFF,    0x7F,    NEXT)
OPDEF(CEE_STSFLD,                     "stsfld",           Pop1,               Push0,
InlineField,        IObjModel,   1,  0xFF,    0x80,    NEXT)
OPDEF(CEE_STOBJ,                      "stobj",            PopI+Pop1,          Push0,
InlineType,         IPrimitive,  1,  0xFF,    0x81,    NEXT)
OPDEF(CEE_CONV_OVF_I1_UN,             "conv.ovf.i1.un",   Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x82,    NEXT)
OPDEF(CEE_CONV_OVF_I2_UN,             "conv.ovf.i2.un",   Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x83,    NEXT)
OPDEF(CEE_CONV_OVF_I4_UN,             "conv.ovf.i4.un",   Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x84,    NEXT)
OPDEF(CEE_CONV_OVF_I8_UN,             "conv.ovf.i8.un",   Pop1,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0x85,    NEXT)
OPDEF(CEE_CONV_OVF_U1_UN,             "conv.ovf.u1.un",   Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x86,    NEXT)
OPDEF(CEE_CONV_OVF_U2_UN,             "conv.ovf.u2.un",   Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x87,    NEXT)
OPDEF(CEE_CONV_OVF_U4_UN,             "conv.ovf.u4.un",   Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x88,    NEXT)
OPDEF(CEE_CONV_OVF_U8_UN,             "conv.ovf.u8.un",   Pop1,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0x89,    NEXT)
OPDEF(CEE_CONV_OVF_I_UN,              "conv.ovf.i.un",    Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x8A,    NEXT)
OPDEF(CEE_CONV_OVF_U_UN,              "conv.ovf.u.un",    Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0x8B,    NEXT)
OPDEF(CEE_BOX,                        "box",              Pop1,               PushRef,
InlineType,         IPrimitive,  1,  0xFF,    0x8C,    NEXT)
OPDEF(CEE_NEWARR,                     "newarr",           PopI,               PushRef,
InlineType,         IObjModel,   1,  0xFF,    0x8D,    NEXT)
OPDEF(CEE_LDLEN,                      "ldlen",            PopRef,             PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x8E,    NEXT)
OPDEF(CEE_LDELEMA,                    "ldelema",          PopRef+PopI,        PushI,
InlineType,         IObjModel,   1,  0xFF,    0x8F,    NEXT)
OPDEF(CEE_LDELEM_I1,                  "ldelem.i1",        PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x90,    NEXT)
OPDEF(CEE_LDELEM_U1,                  "ldelem.u1",        PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x91,    NEXT)
OPDEF(CEE_LDELEM_I2,                  "ldelem.i2",        PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x92,    NEXT)
OPDEF(CEE_LDELEM_U2,                  "ldelem.u2",        PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x93,    NEXT)
OPDEF(CEE_LDELEM_I4,                  "ldelem.i4",        PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x94,    NEXT)
OPDEF(CEE_LDELEM_U4,                  "ldelem.u4",        PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x95,    NEXT)
OPDEF(CEE_LDELEM_I8,                  "ldelem.i8",        PopRef+PopI,        PushI8,
InlineNone,         IObjModel,   1,  0xFF,    0x96,    NEXT)
OPDEF(CEE_LDELEM_I,                   "ldelem.i",         PopRef+PopI,        PushI,
InlineNone,         IObjModel,   1,  0xFF,    0x97,    NEXT)
OPDEF(CEE_LDELEM_R4,                  "ldelem.r4",        PopRef+PopI,        PushR4,
InlineNone,         IObjModel,   1,  0xFF,    0x98,    NEXT)
OPDEF(CEE_LDELEM_R8,                  "ldelem.r8",        PopRef+PopI,        PushR8,
InlineNone,         IObjModel,   1,  0xFF,    0x99,    NEXT)
```

```
OPDEF(CEE_LDELEM_REF,                 "ldelem.ref",       PopRef+PopI,        PushRef,
InlineNone,         IObjModel,   1,  0xFF,    0x9A,    NEXT)
OPDEF(CEE_STELEM_I,                   "stelem.i",         PopRef+PopI+PopI,   Push0,
InlineNone,         IObjModel,   1,  0xFF,    0x9B,    NEXT)
OPDEF(CEE_STELEM_I1,                  "stelem.i1",        PopRef+PopI+PopI,   Push0,
InlineNone,         IObjModel,   1,  0xFF,    0x9C,    NEXT)
OPDEF(CEE_STELEM_I2,                  "stelem.i2",        PopRef+PopI+PopI,   Push0,
InlineNone,         IObjModel,   1,  0xFF,    0x9D,    NEXT)
OPDEF(CEE_STELEM_I4,                  "stelem.i4",        PopRef+PopI+PopI,   Push0,
InlineNone,         IObjModel,   1,  0xFF,    0x9E,    NEXT)
OPDEF(CEE_STELEM_I8,                  "stelem.i8",        PopRef+PopI+PopI8,  Push0,
InlineNone,         IObjModel,   1,  0xFF,    0x9F,    NEXT)
OPDEF(CEE_STELEM_R4,                  "stelem.r4",        PopRef+PopI+PopR4,  Push0,
InlineNone,         IObjModel,   1,  0xFF,    0xA0,    NEXT)
OPDEF(CEE_STELEM_R8,                  "stelem.r8",        PopRef+PopI+PopR8,  Push0,
InlineNone,         IObjModel,   1,  0xFF,    0xA1,    NEXT)
OPDEF(CEE_STELEM_REF,                 "stelem.ref",       PopRef+PopI+PopRef, Push0,
InlineNone,         IObjModel,   1,  0xFF,    0xA2,    NEXT)
OPDEF(CEE_UNUSED2,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA3,    NEXT)
OPDEF(CEE_UNUSED3,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA4,    NEXT)
OPDEF(CEE_UNUSED4,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA5,    NEXT)
OPDEF(CEE_UNUSED5,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA6,    NEXT)
OPDEF(CEE_UNUSED6,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA7,    NEXT)
OPDEF(CEE_UNUSED7,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA8,    NEXT)
OPDEF(CEE_UNUSED8,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xA9,    NEXT)
OPDEF(CEE_UNUSED9,                    "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xAA,    NEXT)
OPDEF(CEE_UNUSED10,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xAB,    NEXT)
OPDEF(CEE_UNUSED11,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xAC,    NEXT)
OPDEF(CEE_UNUSED12,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xAD,    NEXT)
OPDEF(CEE_UNUSED13,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xAE,    NEXT)
OPDEF(CEE_UNUSED14,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xAF,    NEXT)
OPDEF(CEE_UNUSED15,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xB0,    NEXT)
OPDEF(CEE_UNUSED16,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xB1,    NEXT)
OPDEF(CEE_UNUSED17,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xB2,    NEXT)
OPDEF(CEE_CONV_OVF_I1,                "conv.ovf.i1",      Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xB3,    NEXT)
OPDEF(CEE_CONV_OVF_U1,                "conv.ovf.u1",      Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xB4,    NEXT)
OPDEF(CEE_CONV_OVF_I2,                "conv.ovf.i2",      Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xB5,    NEXT)
OPDEF(CEE_CONV_OVF_U2,                "conv.ovf.u2",      Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xB6,    NEXT)
OPDEF(CEE_CONV_OVF_I4,                "conv.ovf.i4",      Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xB7,    NEXT)
OPDEF(CEE_CONV_OVF_U4,                "conv.ovf.u4",      Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xB8,    NEXT)
OPDEF(CEE_CONV_OVF_I8,                "conv.ovf.i8",      Pop1,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0xB9,    NEXT)
OPDEF(CEE_CONV_OVF_U8,                "conv.ovf.u8",      Pop1,               PushI8,
InlineNone,         IPrimitive,  1,  0xFF,    0xBA,    NEXT)
OPDEF(CEE_UNUSED50,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xBB,    NEXT)
OPDEF(CEE_UNUSED18,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xBC,    NEXT)
OPDEF(CEE_UNUSED19,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xBD,    NEXT)
OPDEF(CEE_UNUSED20,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xBE,    NEXT)
OPDEF(CEE_UNUSED21,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xBF,    NEXT)
```

```
OPDEF(CEE_UNUSED22,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC0,    NEXT)
OPDEF(CEE_UNUSED23,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC1,    NEXT)
OPDEF(CEE_REFANYVAL,                  "refanyval",        Pop1,               PushI,
InlineType,         IPrimitive,  1,  0xFF,    0xC2,    NEXT)
OPDEF(CEE_CKFINITE,                  "ckfinite",          Pop1,               PushR8,
InlineNone,         IPrimitive,  1,  0xFF,    0xC3,    NEXT)
OPDEF(CEE_UNUSED24,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC4,    NEXT)
OPDEF(CEE_UNUSED25,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC5,    NEXT)
OPDEF(CEE_MKREFANY,                   "mkrefany",         PopI,               Push1,
InlineType,         IPrimitive,  1,  0xFF,    0xC6,    NEXT)
OPDEF(CEE_UNUSED59,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC7,    NEXT)
OPDEF(CEE_UNUSED60,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC8,    NEXT)
OPDEF(CEE_UNUSED61,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xC9,    NEXT)
OPDEF(CEE_UNUSED62,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xCA,    NEXT)
OPDEF(CEE_UNUSED63,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xCB,    NEXT)
OPDEF(CEE_UNUSED64,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xCC,    NEXT)
OPDEF(CEE_UNUSED65,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xCD,    NEXT)
OPDEF(CEE_UNUSED66,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xCE,    NEXT)
OPDEF(CEE_UNUSED67,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xCF,    NEXT)
OPDEF(CEE_LDTOKEN,                   "ldtoken",           Pop0,               PushI,
InlineTok,          IPrimitive,  1,  0xFF,    0xD0,    NEXT)
OPDEF(CEE_CONV_U2,                    "conv.u2",          Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xD1,    NEXT)
OPDEF(CEE_CONV_U1,                    "conv.u1",          Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xD2,    NEXT)
OPDEF(CEE_CONV_I,                     "conv.i",           Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xD3,    NEXT)
OPDEF(CEE_CONV_OVF_I,                 "conv.ovf.i",       Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xD4,    NEXT)
OPDEF(CEE_CONV_OVF_U,                 "conv.ovf.u",       Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xD5,    NEXT)
OPDEF(CEE_ADD_OVF,                    "add.ovf",          Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0xD6,    NEXT)
OPDEF(CEE_ADD_OVF_UN,                 "add.ovf.un",       Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0xD7,    NEXT)
OPDEF(CEE_MUL_OVF,                    "mul.ovf",          Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0xD8,    NEXT)
OPDEF(CEE_MUL_OVF_UN,                 "mul.ovf.un",       Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0xD9,    NEXT)
OPDEF(CEE_SUB_OVF,                    "sub.ovf",          Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0xDA,    NEXT)
OPDEF(CEE_SUB_OVF_UN,                 "sub.ovf.un",       Pop1+Pop1,          Push1,
InlineNone,         IPrimitive,  1,  0xFF,    0xDB,    NEXT)
OPDEF(CEE_ENDFINALLY,                 "endfinally",       Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xDC,    RETURN)
OPDEF(CEE_LEAVE,                      "leave",            Pop0,               Push0,
InlineBrTarget,     IPrimitive,  1,  0xFF,    0xDD,    BRANCH)
OPDEF(CEE_LEAVE_S,                    "leave.s",          Pop0,               Push0,
ShortInlineBrTarget,IPrimitive,  1,  0xFF,    0xDE,    BRANCH)
OPDEF(CEE_STIND_I,                    "stind.i",          PopI+PopI,          Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xDF,    NEXT)
OPDEF(CEE_CONV_U,                     "conv.u",           Pop1,               PushI,
InlineNone,         IPrimitive,  1,  0xFF,    0xE0,    NEXT)
OPDEF(CEE_UNUSED26,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE1,    NEXT)
OPDEF(CEE_UNUSED27,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE2,    NEXT)
OPDEF(CEE_UNUSED28,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE3,    NEXT)
OPDEF(CEE_UNUSED29,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE4,    NEXT)
OPDEF(CEE_UNUSED30,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE5,    NEXT)
```

```
OPDEF(CEE_UNUSED31,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE6,    NEXT)
OPDEF(CEE_UNUSED32,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE7,    NEXT)
OPDEF(CEE_UNUSED33,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE8,    NEXT)
OPDEF(CEE_UNUSED34,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xE9,    NEXT)
OPDEF(CEE_UNUSED35,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xEA,    NEXT)
OPDEF(CEE_UNUSED36,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xEB,    NEXT)
OPDEF(CEE_UNUSED37,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xEC,    NEXT)
OPDEF(CEE_UNUSED38,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xED,    NEXT)
OPDEF(CEE_UNUSED39,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xEE,    NEXT)
OPDEF(CEE_UNUSED40,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xEF,    NEXT)
OPDEF(CEE_UNUSED41,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF0,    NEXT)
OPDEF(CEE_UNUSED42,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF1,    NEXT)
OPDEF(CEE_UNUSED43,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF2,    NEXT)
OPDEF(CEE_UNUSED44,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF3,    NEXT)
OPDEF(CEE_UNUSED45,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF4,    NEXT)
OPDEF(CEE_UNUSED46,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF5,    NEXT)
OPDEF(CEE_UNUSED47,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF6,    NEXT)
OPDEF(CEE_UNUSED48,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  1,  0xFF,    0xF7,    NEXT)
OPDEF(CEE_PREFIX7,                    "prefix7",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xF8,    META)
OPDEF(CEE_PREFIX6,                    "prefix6",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xF9,    META)
OPDEF(CEE_PREFIX5,                    "prefix5",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xFA,    META)
OPDEF(CEE_PREFIX4,                    "prefix4",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xFB,    META)
OPDEF(CEE_PREFIX3,                    "prefix3",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xFC,    META)
OPDEF(CEE_PREFIX2,                    "prefix2",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xFD,    META)
OPDEF(CEE_PREFIX1,                    "prefix1",          Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xFE,    META)
OPDEF(CEE_PREFIXREF,                  "prefixref",        Pop0,               Push0,
InlineNone,         IInternal,   1,  0xFF,    0xFF,    META)

OPDEF(CEE_ARGLIST,                    "arglist",          Pop0,               PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x00,    NEXT)
OPDEF(CEE_CEQ,                        "ceq",              Pop1+Pop1,          PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x01,    NEXT)
OPDEF(CEE_CGT,                        "cgt",              Pop1+Pop1,          PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x02,    NEXT)
OPDEF(CEE_CGT_UN,                     "cgt.un",           Pop1+Pop1,          PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x03,    NEXT)
OPDEF(CEE_CLT,                        "clt",              Pop1+Pop1,          PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x04,    NEXT)
OPDEF(CEE_CLT_UN,                     "clt.un",           Pop1+Pop1,          PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x05,    NEXT)
OPDEF(CEE_LDFTN,                      "ldftn",            Pop0,               PushI,
InlineMethod,       IPrimitive,  2,  0xFE,    0x06,    NEXT)
OPDEF(CEE_LDVIRTFTN,                  "ldvirtftn",        PopRef,             PushI,
InlineMethod,       IPrimitive,  2,  0xFE,    0x07,    NEXT)
OPDEF(CEE_UNUSED56,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x08,    NEXT)
OPDEF(CEE_LDARG,                      "ldarg",            Pop0,               Push1,
InlineVar,          IPrimitive,  2,  0xFE,    0x09,    NEXT)
OPDEF(CEE_LDARGA,                     "ldarga",           Pop0,               PushI,
InlineVar,          IPrimitive,  2,  0xFE,    0x0A,    NEXT)
```

```
OPDEF(CEE_STARG,                      "starg",            Pop1,               Push0,
InlineVar,          IPrimitive,  2,  0xFE,    0x0B,    NEXT)
OPDEF(CEE_LDLOC,                      "ldloc",            Pop0,               Push1,
InlineVar,          IPrimitive,  2,  0xFE,    0x0C,    NEXT)
OPDEF(CEE_LDLOCA,                     "ldloca",           Pop0,               PushI,
InlineVar,          IPrimitive,  2,  0xFE,    0x0D,    NEXT)
OPDEF(CEE_STLOC,                      "stloc",            Pop1,               Push0,
InlineVar,          IPrimitive,  2,  0xFE,    0x0E,    NEXT)
OPDEF(CEE_LOCALLOC,                   "localloc",         PopI,               PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x0F,    NEXT)
OPDEF(CEE_UNUSED57,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x10,    NEXT)
OPDEF(CEE_ENDFILTER,                  "endfilter",        PopI,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x11,    RETURN)
OPDEF(CEE_UNALIGNED,                  "unaligned.",       Pop0,               Push0,
ShortInlineI,       IPrefix,     2,  0xFE,    0x12,    META)
OPDEF(CEE_VOLATILE,                   "volatile.",        Pop0,               Push0,
InlineNone,         IPrefix,     2,  0xFE,    0x13,    META)
OPDEF(CEE_TAILCALL,                   "tail.",            Pop0,               Push0,
InlineNone,         IPrefix,     2,  0xFE,    0x14,    META)
OPDEF(CEE_INITOBJ,                    "initobj",          PopI,               Push0,
InlineType,         IObjModel,   2,  0xFE,    0x15,    NEXT)
OPDEF(CEE_UNUSED68,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x16,    NEXT)
OPDEF(CEE_CPBLK,                      "cpblk",            PopI+PopI+PopI,     Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x17,    NEXT)
OPDEF(CEE_INITBLK,                    "initblk",          PopI+PopI+PopI,     Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x18,    NEXT)
OPDEF(CEE_UNUSED69,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x19,    NEXT)
OPDEF(CEE_RETHROW,                    "rethrow",          Pop0,               Push0,
InlineNone,         IObjModel,   2,  0xFE,    0x1A,    THROW)
OPDEF(CEE_UNUSED51,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x1B,    NEXT)
OPDEF(CEE_SIZEOF,                     "sizeof",           Pop0,               PushI,
InlineType,         IPrimitive,  2,  0xFE,    0x1C,    NEXT)
OPDEF(CEE_REFANYTYPE,                 "refanytype",       Pop1,               PushI,
InlineNone,         IPrimitive,  2,  0xFE,    0x1D,    NEXT)
OPDEF(CEE_UNUSED52,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x1E,    NEXT)
OPDEF(CEE_UNUSED53,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x1F,    NEXT)
OPDEF(CEE_UNUSED54,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x20,    NEXT)
OPDEF(CEE_UNUSED55,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x21,    NEXT)
OPDEF(CEE_UNUSED70,                   "unused",           Pop0,               Push0,
InlineNone,         IPrimitive,  2,  0xFE,    0x22,    NEXT)

// These are not real opcodes, but they are handy internally in the EE

OPDEF(CEE_ILLEGAL,                    "illegal",          Pop0,               Push0,
InlineNone,         IInternal,   0,  MOOT,    MOOT,    META)
OPDEF(CEE_MACRO_END,                  "endmac",           Pop0,               Push0,
InlineNone,         IInternal,   0,  MOOT,    MOOT,    META)



#ifndef OPALIAS
#define _OPALIAS_DEFINED_
#define OPALIAS(canonicalName, stringName, realOpcode)
#endif

OPALIAS(CEE_BRNULL,        "brnull",            CEE_BRFALSE)
OPALIAS(CEE_BRNULL_S,      "brnull.s",          CEE_BRFALSE_S)
OPALIAS(CEE_BRZERO,        "brzero",            CEE_BRFALSE)
OPALIAS(CEE_BRZERO_S,      "brzero.s",          CEE_BRFALSE_S)
OPALIAS(CEE_BRINST,        "brinst",            CEE_BRTRUE)
OPALIAS(CEE_BRINST_S,      "brinst.s",          CEE_BRTRUE_S)
OPALIAS(CEE_LDIND_U8,      "ldind.u8",          CEE_LDIND_I8)
OPALIAS(CEE_LDELEM_U8,     "ldelem.u8",         CEE_LDELEM_I8)
OPALIAS(CEE_LDC_I4_M1x,    "ldc.i4.M1",         CEE_LDC_I4_M1)
OPALIAS(CEE_ENDFAULT,      "endfault",          CEE_ENDFINALLY)

#ifdef _OPALIAS_DEFINED_
#undef OPALIAS
```

```
#undef _OPALIAS_DEFINED_
#endif
```

C.3 Complete grammar

This grammar provides a number of ease-of-use features not provided in the grammar of Partition II, as well as supporting some features which are not portable across implementations and hence are not part of this standard. Unlike the grammar of Partition II, this one is designed for ease of programming rather than ease of reading; it can be converted directly into a YACC grammar.

```
Lexical tokens
    ID - C style alphaNumeric identifier (e.g., Hello_There2)
    QSTRING  - C style quoted string (e.g.,  "hi\n")
    SQSTRING - C style singly quoted string(e.g.,  'hi')
    INT32    - C style 32-bit integer (e.g.,  235,  03423, 0x34FFF)
    INT64    - C style 64-bit integer (e.g.,  -2353453636235234,
0x34FFFFFFFFFF)
    FLOAT64  - C style floating point number (e.g.,  -0.2323,
354.3423, 3435.34E-5)
    INSTR_*  - IL instructions of a particular class (see
opcode.def).
---------------------------------------------------------------------
-------------
START           : decls
                ;

decls                   : /* EMPTY */
                        | decls decl
                        ;

decl                    : classHead '{' classDecls '}'
                        | nameSpaceHead '{' decls '}'
                        | methodHead  methodDecls '}'
                        | fieldDecl
                        | dataDecl
                        | vtableDecl
                        | vtfixupDecl
                        | extSourceSpec
                        | fileDecl
                        | assemblyHead '{' assemblyDecls '}'
                        | assemblyRefHead '{' assemblyRefDecls '}'
                        | comtypeHead '{' comtypeDecls '}'
                        | manifestResHead '{' manifestResDecls '}'
                        | moduleHead
                        | secDecl
                        | customAttrDecl
                                    | '.subsystem' int32
                                    | '.corflags' int32
                                    | '.file' 'alignment' int32
                                    | '.imagebase' int64
                                    | languageDecl
                        ;

compQstring             : QSTRING
                        | compQstring '+' QSTRING
                                    ;

languageDecl                  : '.language' SQSTRING
                        | '.language' SQSTRING ',' SQSTRING
                        | '.language' SQSTRING ',' SQSTRING ','
SQSTRING
                                    ;

customAttrDecl          : '.custom' customType
                        | '.custom' customType '=' compQstring
                        | customHead bytes ')'
                        | '.custom' '(' ownerType ')' customType
                        | '.custom' '(' ownerType ')' customType '='
compQstring
                        | customHeadWithOwner bytes ')'
                        ;

moduleHead              : '.module'
                        | '.module' name1
                                    | '.module' 'extern' name1
                        ;


vtfixupDecl             : '.vtfixup' '[' int32 ']' vtfixupAttr 'at'
id
                        ;

vtfixupAttr             : /* EMPTY */
                        | vtfixupAttr 'int32'
                        | vtfixupAttr 'int64'
                        | vtfixupAttr 'fromunmanaged'
                        | vtfixupAttr 'callmostderived'
                        ;

vtableDecl              : vtableHead bytes ')'
                        ;

vtableHead              : '.vtable' '=' '('
                        ;

nameSpaceHead           : '.namespace' name1
                        ;

classHead               : '.class' classAttr id extendsClause
implClause
                        ;

classAttr               : /* EMPTY */
                        | classAttr 'public'
                        | classAttr 'private'
                        | classAttr 'value'
                        | classAttr 'enum'
                        | classAttr 'interface'
                        | classAttr 'sealed'
                        | classAttr 'abstract'
                        | classAttr 'auto'
                        | classAttr 'sequential'
                        | classAttr 'explicit'
                        | classAttr 'ansi'
                        | classAttr 'unicode'
                        | classAttr 'autochar'
                        | classAttr 'import'
                        | classAttr 'serializable'
                        | classAttr 'nested' 'public'
                        | classAttr 'nested' 'private'
                        | classAttr 'nested' 'family'
                        | classAttr 'nested' 'assembly'
                        | classAttr 'nested' 'famandassem'
                        | classAttr 'nested' 'famorassem'
                        | classAttr 'beforefieldinit'
                        | classAttr 'specialname'
                        | classAttr 'rtspecialname'
                        ;

extendsClause           : /* EMPTY */
                        | 'extends' className
                        ;

implClause              : /* EMPTY */
                        | 'implements' classNames
                                                ;

classNames              : classNames ',' className
                        | className
                        ;

classDecls              : /* EMPTY */
                        | classDecls classDecl
                        ;

classDecl               : methodHead  methodDecls '}'
                        | classHead '{' classDecls '}'
                        | eventHead '{' eventDecls '}'
                        | propHead '{' propDecls '}'
                        | fieldDecl
                        | dataDecl
                        | secDecl
                        | extSourceSpec
                        | customAttrDecl
                        | '.size' int32
                        | '.pack' int32
                        | exportHead '{' comtypeDecls '}'
                        | '.override' typeSpec '::' methodName 'with'
callConv type typeSpec '::' methodName '(' sigArgs0 ')'
                                    | languageDecl
                        ;

fieldDecl               : '.field' repeatOpt fieldAttr type id atOpt
initOpt
                        ;


atOpt                   : /* EMPTY */
                        | 'at' id
                        ;

initOpt                 : /* EMPTY */
                        | '=' fieldInit
                                    ;

repeatOpt                     : /* EMPTY */
                        | '[' int32 ']'
                                    ;

customHead              : '.custom' customType '=' '('
                        ;
```

```
customHeadWithOwner     : '.custom' '(' ownerType ')' customType '='
'('
                        ;

memberRef                     : methodSpec callConv type typeSpec
'::' methodName '(' sigArgs0 ')'
                        | methodSpec callConv type methodName '('
sigArgs0 ')'
                        | 'field' type typeSpec '::' id
                        | 'field' type id
                        ;

customType              : callConv type typeSpec '::' '.ctor' '('
sigArgs0 ')'
                        | callConv type '.ctor' '(' sigArgs0 ')'
                        ;

ownerType               : typeSpec
                        | memberRef
                        ;

eventHead               : '.event' eventAttr typeSpec id
                        | '.event' eventAttr id
                        ;


eventAttr               : /* EMPTY */
                        | eventAttr 'rtspecialname' /**/
                        | eventAttr 'specialname'
                        ;

eventDecls              : /* EMPTY */
                        | eventDecls eventDecl
                        ;

eventDecl               : '.addon' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.addon' callConv type methodName '('
sigArgs0 ')'
                        | '.removeon' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.removeon' callConv type methodName '('
sigArgs0 ')'
                        | '.fire' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.fire' callConv type methodName '('
sigArgs0 ')'
                        | '.other' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.other' callConv type methodName '('
sigArgs0 ')'
                        | extSourceSpec
                        | customAttrDecl
                                    | languageDecl
                        ;

propHead                : '.property' propAttr callConv type id '('
sigArgs0 ')' initOpt
                        ;
```

```
propAttr                : /* EMPTY */
                        | propAttr 'rtspecialname' /**/
                        | propAttr 'specialname'
                        ;

propDecls               : /* EMPTY */
                        | propDecls propDecl
                        ;


propDecl                : '.set' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.set' callConv type methodName '('
sigArgs0 ')'
                        | '.get' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.get' callConv type methodName '('
sigArgs0 ')'
                        | '.other' callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | '.other' callConv type methodName '('
sigArgs0 ')'
                        | customAttrDecl
                        | extSourceSpec
                                    | languageDecl
                        ;


methodHeadPart1         : '.method'
                        ;

methodHead              : methodHeadPart1 methAttr callConv paramAttr
type methodName '(' sigArgs0 ')' implAttr '{'
                        | methodHeadPart1 methAttr callConv paramAttr
type 'marshal' '(' nativeType ')' methodName '(' sigArgs0 ')'
implAttr '{'
                        ;


methAttr                : /* EMPTY */

                        | methAttr 'static'
                        | methAttr 'public'
                        | methAttr 'private'
                        | methAttr 'family'
                        | methAttr 'final'
                        | methAttr 'specialname'
                        | methAttr 'virtual'
                        | methAttr 'abstract'
                        | methAttr 'assembly'
                        | methAttr 'famandassem'
                        | methAttr 'famorassem'
                        | methAttr 'privatescope'
                        | methAttr 'hidebysig'
                        | methAttr 'newslot'
                        | methAttr 'rtspecialname' /**/
                        | methAttr 'unmanagedexp'
                        | methAttr 'reqsecobj'

                        | methAttr 'pinvokeimpl' '(' compQstring 'as'
compQstring pinvAttr ')'
```

```
                        | methAttr 'pinvokeimpl' '(' compQstring
pinvAttr ')'
                        | methAttr 'pinvokeimpl' '(' pinvAttr ')'
                        ;

pinvAttr                : /* EMPTY */
                        | pinvAttr 'nomangle'
                        | pinvAttr 'ansi'
                        | pinvAttr 'unicode'
                        | pinvAttr 'autochar'
                        | pinvAttr 'lasterr'
                        | pinvAttr 'winapi'
                        | pinvAttr 'cdecl'
                        | pinvAttr 'stdcall'
                        | pinvAttr 'thiscall'
                        | pinvAttr 'fastcall'
                        ;

methodName              : '.ctor'
                        | '.cctor'
                        | name1
                        ;

paramAttr               : /* EMPTY */
                        | paramAttr '[' 'in' ']'
                        | paramAttr '[' 'out' ']'
                        | paramAttr '[' 'opt' ']'
                        | paramAttr '[' int32 ']'
                        ;

fieldAttr               : /* EMPTY */
                        | fieldAttr 'static'
                        | fieldAttr 'public'
                        | fieldAttr 'private'
                        | fieldAttr 'family'
                        | fieldAttr 'initonly'
                        | fieldAttr 'rtspecialname'  /**/
                        | fieldAttr 'specialname'
                                    /* commented out because PInvoke
for fields is not supported by EE
                        | fieldAttr 'pinvokeimpl' '(' compQstring
'as' compQstring pinvAttr ')'
                        | fieldAttr 'pinvokeimpl' '(' compQstring
pinvAttr ')'
                        | fieldAttr 'pinvokeimpl' '(' pinvAttr ')'
                                    */
                        | fieldAttr 'marshal' '(' nativeType ')'
                        | fieldAttr 'assembly'
                        | fieldAttr 'famandassem'
                        | fieldAttr 'famorassem'
                        | fieldAttr 'privatescope'
                        | fieldAttr 'literal'
                        | fieldAttr 'notserialized'
                        ;

implAttr                : /* EMPTY */
                        | implAttr 'native'
                        | implAttr 'cil'
                        | implAttr 'optil'
                        | implAttr 'managed'
                        | implAttr 'unmanaged'
                        | implAttr 'forwardref'
                        | implAttr 'preservesig'
                        | implAttr 'runtime'
                        | implAttr 'internalcall'
                        | implAttr 'synchronized'
                        | implAttr 'noinlining'
                        ;

localsHead              : '.locals'
                        ;


methodDecl              : '.emitbyte' int32
                        | sehBlock
                        | '.maxstack' int32
                        | localsHead '(' sigArgs0 ')'
                        | localsHead 'init' '(' sigArgs0 ')'
                        | '.entrypoint'
                        | '.zeroinit'
                        | dataDecl
                        | instr
                        | id ':'
                        | secDecl
                        | extSourceSpec
                                    | languageDecl
                        | customAttrDecl
                                    | '.export' '[' int32 ']'
                                    | '.export' '[' int32 ']'
      'as' id
                        | '.vtentry' int32 ':' int32
                        | '.override' typeSpec '::' methodName
                        | scopeBlock
                        | '.param' '[' int32 ']' initOpt
                        ;

scopeBlock              : scopeOpen methodDecls '}'
                        ;

scopeOpen               : '{'
                        ;

sehBlock                : tryBlock sehClauses
                        ;

sehClauses              : sehClause sehClauses
                        | sehClause
                        ;

tryBlock                : tryHead scopeBlock
                        | tryHead id 'to' id
                        | tryHead int32 'to' int32
                        ;

tryHead                 : '.try'
                        ;


sehClause               : catchClause handlerBlock
                        | filterClause handlerBlock
                        | finallyClause handlerBlock
                        | faultClause handlerBlock
                        ;

filterClause            : filterHead scopeBlock
                        | filterHead id
                        | filterHead int32
                        ;

filterHead              : 'filter'
                        ;

catchClause             : 'catch' className
                        ;

finallyClause           : 'finally'
                        ;

faultClause             : 'fault'
                        ;

handlerBlock            : scopeBlock
                        | 'handler' id 'to' id
                        | 'handler' int32 'to' int32
                        ;


methodDecls             : /* EMPTY */
                        | methodDecls methodDecl
                        ;

dataDecl                : ddHead ddBody
                        ;

ddHead                  : '.data' tls id '='
                        | '.data' tls
                        ;

tls                     : /* EMPTY */
                        | 'tls'
                        ;

ddBody                  : '{' ddItemList '}'
                        | ddItem
                        ;

ddItemList              : ddItem ',' ddItemList
                        | ddItem
                        ;

ddItemCount             : /* EMPTY */
                        | '[' int32 ']'
                        ;

ddItem                  : 'char' '*' '(' compQstring ')'
                        | '&' '(' id ')'
                        | bytearrayhead bytes ')'
                        | 'float32' '(' float64 ')' ddItemCount
                        | 'float64' '(' float64 ')' ddItemCount
                        | 'int64' '(' int64 ')' ddItemCount
                        | 'int32' '(' int32 ')' ddItemCount
                        | 'int16' '(' int32 ')' ddItemCount
                        | 'int8' '(' int32 ')' ddItemCount
                        | 'float32' ddItemCount
                        | 'float64' ddItemCount
                        | 'int64' ddItemCount
                        | 'int32' ddItemCount
                        | 'int16' ddItemCount
                        | 'int8' ddItemCount
                        ;

fieldInit               : 'float32' '(' float64 ')'
                        | 'float64' '(' float64 ')'
                        | 'float32' '(' int64 ')'
                        | 'float64' '(' int64 ')'
                        | 'int64' '(' int64 ')'
                        | 'int32' '(' int64 ')'
                        | 'int16' '(' int64 ')'
                        | 'char' '(' int64 ')'
                        | 'int8' '(' int64 ')'
                        | 'bool' '(' truefalse ')'
                        | compQstring
                        | bytearrayhead bytes ')'
                                    | 'nullref'
                        ;

bytearrayhead           : 'bytearray' '('
                        ;

bytes                         : /* EMPTY */
                                    | hexbytes
                                    ;

hexbytes                : HEXBYTE
                        | hexbytes HEXBYTE
                        ;

instr_r_head            : INSTR_R '('
                        ;

instr_tok_head          : INSTR_TOK
                        ;

methodSpec              : 'method'
                        ;

instr                   : INSTR_NONE
                        | INSTR_VAR int32
                        | INSTR_VAR id
                        | INSTR_I int32
                        | INSTR_I8 int64
                        | INSTR_R float64
                        | INSTR_R int64
                        | instr_r_head bytes ')'
                        | INSTR_BRTARGET int32
                        | INSTR_BRTARGET id
                        | INSTR_METHOD callConv type typeSpec '::'
methodName '(' sigArgs0 ')'
                        | INSTR_METHOD callConv type methodName '('
sigArgs0 ')'
                        | INSTR_FIELD type typeSpec '::' id
                        | INSTR_FIELD type id
                        | INSTR_TYPE typeSpec
                        | INSTR_STRING compQstring
                        | INSTR_STRING bytearrayhead bytes ')'
                        | INSTR_SIG callConv type '(' sigArgs0 ')'
                        | INSTR_RVA id
                        | INSTR_RVA int32
                        | instr_tok_head ownerType /* ownerType ::=
memberRef | typeSpec */
                        | INSTR_SWITCH '(' labels ')'
                        | INSTR_PHI int16s
                        ;

sigArgs0                : /* EMPTY */
                        | sigArgs1
                        ;

sigArgs1                : sigArg
                        | sigArgs1 ',' sigArg
                        ;

sigArg                  : '...'
                        | paramAttr type
                        | paramAttr type id
                        | paramAttr type 'marshal' '(' nativeType ')'
                        | paramAttr type 'marshal' '(' nativeType ')'
id
                        ;

name1                   : id
                        | DOTTEDNAME
                        | name1 '.' name1
                        ;

className               : '[' name1 ']' slashedName
                        | '[' '.module' name1 ']' slashedName
                        | slashedName
                        ;

slashedName             : name1
                        | slashedName '/' name1
                        ;

typeSpec                : className
                        | '[' name1 ']'
                        | '[' '.module' name1 ']'
                        | type
                        ;

callConv                : 'instance' callConv
                        | 'explicit' callConv
                        | callKind
                        ;

callKind                : /* EMPTY */
                        | 'default'
                        | 'vararg'
                        | 'unmanaged' 'cdecl'
                        | 'unmanaged' 'stdcall'
                        | 'unmanaged' 'thiscall'
                        | 'unmanaged' 'fastcall'
                        ;
```

```
nativeType               : /* EMPTY */
                         | 'custom' '(' compQstring ',' compQstring
',' compQstring ',' compQstring ')'
                         | 'custom' '(' compQstring ',' compQstring
')'
                         | 'fixed' 'sysstring' '[' int32 ']'
                         | 'fixed' 'array' '[' int32 ']'
                         | 'variant'
                         | 'currency'
                         | 'syschar'
                         | 'void'
                         | 'bool'
                         | 'int8'
                         | 'int16'
                         | 'int32'
                         | 'int64'
                         | 'float32'
                         | 'float64'
                         | 'error'
                         | 'unsigned' 'int8'
                         | 'unsigned' 'int16'
                         | 'unsigned' 'int32'
                         | 'unsigned' 'int64'
                         | nativeType '*'
                         | nativeType '[' ']'
                         | nativeType '[' int32 ']'
                         | nativeType '[' int32 '+' int32 ']'
                         | nativeType '[' '+' int32 ']'
                                     | 'decimal'
                         | 'date'
                         | 'bstr'
                         | 'lpstr'
                         | 'lpwstr'
                         | 'lptstr'
                         | 'objectref'
                         | 'iunknown'
                         | 'idispatch'
                         | 'struct'
                         | 'interface'
                         | 'safearray' variantType
                         | 'safearray' variantType ',' compQstring

                         | 'int'
                         | 'unsigned' 'int'
                         | 'nested' 'struct'
                         | 'byvalstr'
                         | 'ansi' 'bstr'
                         | 'tbstr'
                         | 'variant' 'bool'
                         | methodSpec
                         | 'as' 'any'
                         | 'lpstruct'
                         ;

variantType              : /* EMPTY */
                         | 'null'
                         | 'variant'
                         | 'currency'
                         | 'void'
                         | 'bool'
                         | 'int8'
```

```
                        | 'int16'
                        | 'int32'
                        | 'int64'
                        | 'float32'
                        | 'float64'
                        | 'unsigned' 'int8'
                        | 'unsigned' 'int16'
                        | 'unsigned' 'int32'
                        | 'unsigned' 'int64'
                        | '*'
                        | variantType '[' ']'
                        | variantType 'vector'
                        | variantType '&'
                        | 'decimal'
                        | 'date'
                        | 'bstr'
                        | 'lpstr'
                        | 'lpwstr'
                        | 'iunknown'
                        | 'idispatch'
                        | 'safearray'
                        | 'int'
                        | 'unsigned' 'int'
                        | 'error'
                        | 'hresult'
                        | 'carray'
                        | 'userdefined'
                        | 'record'
                        | 'filetime'
                        | 'blob'
                        | 'stream'
                        | 'storage'
                        | 'streamed_object'
                        | 'stored_object'
                        | 'blob_object'
                        | 'cf'
                        | 'clsid'
                        ;

type                    : 'class' className
                                    | 'object'
                                    | 'string'
                        | 'value' 'class' className
                        | 'valuetype' className
                        | type '[' ']'
                        | type '[' bounds1 ']'
                                    /* uncomment when and if this
type is supported by the Runtime
                        | type 'value' '[' int32 ']'
                        */
                                    | type '&'
                        | type '*'
                        | type 'pinned'
                        | type 'modreq' '(' className ')'
                        | type 'modopt' '(' className ')'
                        | '!' int32
                        | methodSpec callConv type '*' '(' sigArgs0
')'
                        | 'typedref'
                        | 'char'
                        | 'void'
```

```
                         | 'bool'
                         | 'int8'
                         | 'int16'
                         | 'int32'
                         | 'int64'
                         | 'float32'
                         | 'float64'
                         | 'unsigned' 'int8'
                         | 'unsigned' 'int16'
                         | 'unsigned' 'int32'
                         | 'unsigned' 'int64'
                         | 'native' 'int'
                         | 'native' 'unsigned' 'int'
                         | 'native' 'float'
                         ;

bounds1                  : bound
                         | bounds1 ',' bound
                         ;

bound                    : /* EMPTY */
                         | '...'
                         | int32
                         | int32 '...' int32
                         | int32 '...'
                         ;

labels                   : /* empty */
                         | id ',' labels
                         | int32 ',' labels
                         | id
                         | int32
                         ;


id                       : ID
                         | SQSTRING
                         ;

int16s                   : /* EMPTY */
                         | int16s int32
                         ;

int32                    : INT64
                         ;

int64                    : INT64
                         ;

float64                  : FLOAT64
                         | 'float32' '(' int32 ')'
                         | 'float64' '(' int64 ')'
                         ;

secDecl                  : '.permission' secAction typeSpec '('
nameValPairs ')'

                         | '.permission' secAction typeSpec
                         | psetHead bytes ')'
                         ;

psetHead                 : '.permissionset' secAction '=' '('
                         ;

nameValPairs            : nameValPair
                         | nameValPair ',' nameValPairs
                         ;

nameValPair             : compQstring '=' caValue
                         ;

truefalse                     : 'true'
                                    | 'false'
                                    ;

caValue                 : truefalse
                         | int32
                         | 'int32' '(' int32 ')'
                         | compQstring
                         | className '(' 'int8' ':' int32 ')'
                         | className '(' 'int16' ':' int32 ')'
                         | className '(' 'int32' ':' int32 ')'
                         | className '(' int32 ')'
                         ;

secAction               : 'request'
                         | 'demand'
                         | 'assert'
                         | 'deny'
                         | 'permitonly'
                         | 'linkcheck'
                         | 'inheritcheck'
                         | 'reqmin'
                         | 'reqopt'
                         | 'reqrefuse'
                         | 'prejitgrant'
                         | 'prejitdeny'
                         | 'noncasdemand'
                         | 'noncaslinkdemand'
                         | 'noncasinheritance'
                         ;

extSourceSpec           : '.line' int32 SQSTRING
                         | '.line' int32
                         | '.line' int32 ':' int32 SQSTRING
                         | '.line' int32 ':' int32
                         | P_LINE int32 QSTRING
                         ;

fileDecl                : '.file' fileAttr name1 fileEntry hashHead
bytes ')' fileEntry
                         | '.file' fileAttr name1 fileEntry
                         ;

fileAttr                : /* EMPTY */
                         | fileAttr 'nometadata'
                         ;

fileEntry               : /* EMPTY */
                         | '.entrypoint'
                         ;

hashHead                : '.hash' '=' '('
                         ;

assemblyHead            : '.assembly' asmAttr name1
                         ;

asmAttr                 : /* EMPTY */
                         | asmAttr 'noappdomain'
                         | asmAttr 'noprocess'
                         | asmAttr 'nomachine'
                         ;

assemblyDecls           : /* EMPTY */
                         | assemblyDecls assemblyDecl
                         ;

assemblyDecl            : '.hash' 'algorithm' int32
                         | secDecl
                         | asmOrRefDecl

                         ;

asmOrRefDecl            : publicKeyHead bytes ')'
                         | '.ver' int32 ':' int32 ':' int32 ':' int32
                         | '.locale' compQstring
                         | localeHead bytes ')'
                         | customAttrDecl
                         ;

publicKeyHead           : '.publickey' '=' '('
                         ;

publicKeyTokenHead      : '.publickeytoken' '=' '('
                         ;

localeHead              : '.locale' '=' '('
                         ;

assemblyRefHead         : '.assembly' 'extern' name1
                         | '.assembly' 'extern' name1 'as' name1
                         ;

assemblyRefDecls        : /* EMPTY */
                         | assemblyRefDecls assemblyRefDecl
                         ;

assemblyRefDecl         : hashHead bytes ')'
                         | asmOrRefDecl
                         | publicKeyTokenHead bytes ')'
                         ;

comtypeHead             : '.class' 'extern' comtAttr name1
                         ;

exportHead              : '.export' comtAttr name1
                         ;

comtAttr                : /* EMPTY */
                         | comtAttr 'private'
                         | comtAttr 'public'
                         | comtAttr 'nested' 'public'
                         | comtAttr 'nested' 'private'
```

```
                         | comtAttr 'nested' 'family'
                         | comtAttr 'nested' 'assembly'
                         | comtAttr 'nested' 'famandassem'
                         | comtAttr 'nested' 'famorassem'
                         ;

comtypeDecls             : /* EMPTY */
                         | comtypeDecls comtypeDecl
                         ;

comtypeDecl              : '.file' name1
                         | '.class' 'extern' name1
                         | '.class'  int32
                         | customAttrDecl
                         ;

manifestResHead          : '.mresource' manresAttr name1
                         ;

manresAttr               : /* EMPTY */
                         | manresAttr 'public'
                         | manresAttr 'private'
                         ;

manifestResDecls         : /* EMPTY */
                         | manifestResDecls manifestResDecl
                         ;

manifestResDecl          : '.file' name1 'at' int32
                         | '.assembly' 'extern' name1
                         | customAttrDecl
                         ;
```

C.4 Instruction syntax

While each subclause specifies the exact list of instructions that are included in a grammar class, this information is subject to change over time. The precise format of an instruction can be found by combining the information in §C.1 with the information in the following table:

Table 1: Instruction Syntax classes

Grammar Class	Format(s) Specified in §C.1
`<instr_brtarget>`	`InlineBrTarget, ShortInlineBrTarget`
`<instr_field>`	`InlineField`
`<instr_i>`	`InlineI, ShortInlineI`
`<instr_i8>`	`InlineI8`
`<instr_method>`	`InlineMethod`
`<instr_none>`	`InlineNone`
`<instr_phi>`	`InlinePhi`
`<instr_r>`	`InlineR, ShortInlineR`
`<instr_rva>`	`InlineRVA`
`<instr_sig>`	`InlineSig`
`<instr_string>`	`InlineString`

`<instr_switch>`	`InlineSwitch`
`<instr_tok>`	`InlineTok`
`<instr_type>`	`InlineType`
`<instr_var>`	`InlineVar, ShortInlineVar`

C.4.1 Top-level instruction syntax

```
<instr> ::=
    <instr_brtarget> <int32>
  | <instr_brtarget> <label>
  | <instr_field> <type> [ <typeSpec> :: ] <id>
  | <instr_i> <int32>
  | <instr_i8> <int64>
  | <instr_method>
      <callConv> <type> [ <typeSpec> :: ]
        <methodName> ( <parameters> )
  | <instr_none>
  | <instr_phi> <int16>*
  | <instr_r> ( <bytes> ) // <bytes> represent the binary image
of
                          // float or double (4 or 8 bytes,
                          // respectively)
  | <instr_r> <float64>
  | <instr_r> <int64>     // integer is converted to float
                          // with possible
                          // loss of precision
  | <instr_sig> <callConv> <type> ( <parameters> )
  | <instr_string> bytearray ( <bytes> )
  | <instr_string> <QSTRING>
  | <instr_switch> ( <labels> )
  | <instr_tok> field <type> [ <typeSpec> :: ] <id>
  | <instr_tok> b
      <callConv> <type> [ <typeSpec> :: ]
        <methodName> ( <parameters> )
  | <instr_tok> <typeSpec>
  | <instr_type> <typeSpec>
  | <instr_var> <int32>
  | <instr_var> <localname>
```

C.4.2 Instructions with no operand

These instructions require no operands, so they simply appear by themselves.

```
<instr> ::= <instr_none>
<instr_none> ::= // Derived from opcode.def
       add            | add.ovf     | add.ovf.un     | and
    |
       arglist        | break       | ceq            | cgt
    |
       cgt.un         | ckfinite    | clt            | clt.un
    |
       conv.i         | conv.i1     | conv.i2        | conv.i4
    |
       conv.i8        | conv.ovf.i  | conv.ovf.i.un  |
    conv.ovf.i1|
       conv.ovf.i1.un | conv.ovf.i2 | conv.ovf.i2.un |
    conv.ovf.i4|
       conv.ovf.i4.un | conv.ovf.i8 | conv.ovf.i8.un |
    conv.ovf.u |
       conv.ovf.u.un  | conv.ovf.u1 | conv.ovf.u1.un |
    conv.ovf.u2|
       conv.ovf.u2.un | conv.ovf.u4 | conv.ovf.u4.un |
    conv.ovf.u8|
       conv.ovf.u8.un | conv.r.un   | conv.r4        | conv.r8
    |
       conv.u         | conv.u1     | conv.u2        | conv.u4
    |
       conv.u8        | cpblk       | div            | div.un
    |
       dup            | endfault    | endfilter      |
    endfinally |
       initblk        |             | ldarg.0        | ldarg.1
    |
       ldarg.2        | ldarg.3     | ldc.i4.0       | ldc.i4.1
    |
       ldc.i4.2       | ldc.i4.3    | ldc.i4.4       | ldc.i4.5
    |
       ldc.i4.6       | ldc.i4.7    | ldc.i4.8       | ldc.i4.M1
    |
       ldelem.i       | ldelem.i1   | ldelem.i2      | ldelem.i4
    |
       ldelem.i8      | ldelem.r4   | ldelem.r8      |
    ldelem.ref |
       ldelem.u1      | ldelem.u2   | ldelem.u4      | ldind.i
    |
       ldind.i1       | ldind.i2    | ldind.i4       | ldind.i8
    |
       ldind.r4       | ldind.r8    | ldind.ref      | ldind.u1
    |
       ldind.u2       | ldind.u4    | ldlen          | ldloc.0
    |
       ldloc.1        | ldloc.2     | ldloc.3        | ldnull
    |
       localloc       | mul         | mul.ovf        |
    mul.ovf.un |
       neg            | nop         | not            | or
    |
       pop            | refanytype  | rem            | rem.un
    |
       ret            | rethrow     | shl            | shr
    |
       shr.un         | stelem.i    | stelem.i1      | stelem.i2
    |
       stelem.i4      | stelem.i8   | stelem.r4      | stelem.r8
    |
       stelem.ref     | stind.i     | stind.i1       | stind.i2
    |
       stind.i4       | stind.i8    | stind.r4       | stind.r8
    |
       stind.ref      | stloc.0     | stloc.1        | stloc.2
    |
       stloc.3        | sub         | sub.ovf        |
    sub.ovf.un |
       tail.          | throw       | volatile.      | xor
```

Examples:

```
ldlen
not
```

C.4.3 Instructions that refer to parameters or local variables

These instructions take one operand, which references a parameter or local variable of the current method. The variable can be referenced by its number (starting with variable 0) or by name (if the names are supplied as part of a signature using the form that supplies both a type and a name).

```
<instr> ::= <instr_var> <int32> |
            <instr_var> <localname>
<instr_var> ::= // Derived from opcode.def
               | ldarg    | ldarg.s  | ldarga
    ldarga.s   | ldloc    | ldloc.s  | ldloca
    ldloca.s   | starg    | starg.s  | stloc
    stloc.s
```

Examples:

```
stloc 0          // store into 0th local
ldarg X3      // load from argument named X3
```

C.4.4 Instructions that take a single 32-bit integer argument

These instructions take one operand, which must be a 32-bit integer.

```
<instr> ::= <instr_i> <int32>
<instr_i> ::= // Derived from opcode.def
    ldc.i4 | ldc.i4.s | unaligned.
```

Examples:

```
    ldc.i4 123456  // Load the number 123456
    ldc.i4.s 10    // Load the number 10
```

C.4.5 Instructions that take a single 64-bit integer argument

These instructions take one operand, which must be a 64-bit integer.

```
<instr> ::= <instr_i8> <int64>
<instr_i8> ::= // Derived from opcode.def
    ldc.i8
```

Examples:

```
    ldc.i8 0x123456789AB
    ldc.i8 12
```

C.4.6 Instructions that take a single floating-point argument

These instructions take one operand, which must be a floating point number.

```
    <instr> ::= <instr_r> <float64> |
                <instr_r> <int64>   |
                          <instr_r> ( <bytes> )  // <bytes> is
    binary image
<instr_r> ::= // Derived from opcode.def
ldc.r4 | ldc.r8
```

Examples:

```
    ldc.r4 10.2
    ldc.r4 10
    ldc.r4 0x123456789ABCDEF
    ldc.r8 (00 00 00 00 00 00 F8 FF)
```

C.4.7 Branch instructions

The assembler does not optimize branches. The branch must be specified explicitly as using either the short or long form of the instruction. If the displacement is too large for the short form, then the assembler will display an error.

```
<instr> ::=
    <instr_brtarget> <int32> |
    <instr_brtarget> <label>
<instr_brtarget> ::= // Derived from opcode.def
                          | beq    | beq.s    | bge    | bge.s
    |
    bge.un    | bge.un.s   | bgt    | bgt.s    | bgt.un |
    bgt.un.s |
    ble       | ble.s      | ble.un | ble.un.s | blt    | blt.s
    |
    blt.un    | blt.un.s   | bne.un | bne.un.s | br     | br.s
    |
    brfalse   | brfalse.s  | brtrue | brtrue.s | leave  |
    leave.s
```

Example:

```
br.s 22
br foo
```

C.4.8 Instructions that take a method as an argument

These instructions reference a method, either in another class (first instruction format) or in the current class (second instruction format).

```
<instr> ::=
   <instr_method>
     <callConv> <type> [ <typeSpec> :: ] <methodName> (
<parameters> )
<instr_method> ::= // Derived from opcode.def
     call  | callvirt | jmp | ldftn    | ldvirtftn        |
newobj
```

Examples:

```
call instance int32 C.D.E::X(class W, native int)
ldftn vararg char F(...)    // Global Function F
```

C.4.9 Instructions that take a field of a class as an argument

These instructions reference a field of a class.

```
<instr> ::=
    <instr_field> <type> <typeSpec> :: <id>
<instr_field> ::= // Derived from opcode.def
    ldfld | ldflda | ldsfld | ldsflda | stfld | stsfld
```

Examples:

```
ldfld native int X::IntField
stsfld int32 Y::AnotherField
```

C.4.10 Instructions that take a type as an argument

These instructions reference a type.

```
<instr> ::= <instr_type> <typeSpec>
<instr_type> ::= // Derived from opcode.def
    box     | castclass | cpobj    | initobj | isinst    |
    ldelema | ldobj     | mkrefany | newarr  | refanyval |
    sizeof  | stobj     | unbox
```

Examples:

```
    initobj [mscorlib]System.Console
    sizeof class X
```

C.4.11 Instructions that take a string as an argument

These instructions take a string as an argument.

```
<instr> ::= <instr_string> <QSTRING>
<instr_string> ::= // Derived from opcode.def
    ldstr
```

Examples:

```
    ldstr “This is a string”
    ldstr “This has a\nnewline in it”
```

C.4.12 Instructions that take a signature as an argument

These instructions take a stand-alone signature as an argument.

```
<instr> ::= <instr_sig> <callConv> <type> ( <parameters> )
<instr_sig> ::= // Derived from opcode.def
    calli
```

Examples:

```
    calli class A.B(wchar *)
    calli vararg bool(int32[,] X, ...)
    // Returns a boolean, takes at least one argument. The first
    // argument, named X, must be a two-dimensional array of
    // 32-bit ints
```

C.4.13 Instructions that take a metadata token as an argument

This instruction takes a metadata token as an argument. The token can reference a type, a method, or a field of a class.

```
<instr> ::= <instr_tok> <typeSpec> |
            <instr_tok> method
               <callConv> <type> <typeSpec> :: <methodName>
                          ( <parameters> ) |
            <instr_tok> method
               <callConv> <type> <methodName>
                          ( <parameters> ) |
            <instr_tok> field <type> <typeSpec> :: <id>
<instr_tok> ::= // Derived from opcode.def
    ldtoken
```

Examples:

```
    ldtoken class [mscorlib]System.Console
    ldtoken method int32 X::Fn()
    ldtoken method bool GlobalFn(int32 &)
    ldtoken field class X.Y Class::Field
```

C.4.14 Switch instruction

The switch instruction takes a set of labels or decimal relative values.

```
<instr> ::= <instr_switch> ( <labels> )
<instr_switch> ::= // Derived from opcode.def
    switch
```

Examples:

```
    switch (0x3, -14, Label1)
    switch (5, Label2)
```

Annex D Class library design guidelines

This clause contains only informative text

Information on this topic can be found at the following location: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

Annex E Portability considerations

This clause gathers together information about areas where this Standard deliberately leaves leeway to implementations. This leeway is intended to allow compliant implementations to make choices that provide better performance or add value in other ways. But this leeway inherently makes programs non-portable. This clause describes the techniques that can be used to ensure that programs operate the same way independent of the particular implementation of the CLI.

Note that code can be portable even though the data is not, both due to size of integer type and direction of bytes in words. Read/write invariance holds provided the read method corresponds to the write method (i.e., write as int read as int works, but write as string read as int might not).

E.1 Uncontrollable behavior

The following aspects of program behavior are implementation dependent. Many of these items will be familiar to programmers used to writing code designed for portability (for example, the fact that the CLI does not impose a minimum size for heap or stack).

1. Size of heap and stack aren't required to have minimum sizes
2. Behavior relative to asynchronous exceptions (see `System.Thread.Abort`)
3. Globalization is not supported, so every implementation specifies its culture information including such user-visible features as sort order for strings.
4. Threads cannot be assumed to be either pre-emptively or non-pre-emptively scheduled. This decision is implementation specific.
5. Locating assemblies is an implementation-specific mechanism.
6. Security policy is an implemenation-specific mechanism.
7. File names are implementation-specific.
8. Timer resolution (granularity) is implementation-specific, although the unit is specified.

E.2 Language- and compiler-controllable behavior

The following aspects of program behavior can be controlled through language design or careful generation of CIL by a language-specific compiler. The CLI provides all the support necessary to control the behavior, but the default is to allow implementation-specific optimizations.

1. Unverifiable code can access arbitrary memory and cannot be guaranteed to be portable
2. Floating point – compiler can force all intermediate values to known precision
3. Integer overflow – compiler can force overflow checking
4. Native integer type need not be exposed, or can be exposed for opaque handles only, or can reliably recast with overflow check to known size values before use. Note that "free conversion" between native integer and fixed-size integer without overflow checks will not be portable.
5. Deterministic initialization of types is portable, but "before first reference to static variable" is not. Language design can either force all initialization to be deterministic (cf. Java) or can restrict initialization to deterministic cases (i.e., simple static assignments).

E.3 Programmer-controllable behavior

The following aspects of program behavior can be controlled directly by the programmer.

1. Code that is not thread-safe might operate differently even on a single implementation. In particular, the atomicity guarantees around 64-bit must be adhered to and testing on 64-bit implementations might not be sufficient to find all

such problems. The key is never to use both normal read/write and interlocked access to the same 64-bit datum.

2. Calls to unmanaged code or calls to non-standardized extensions to libraries
3. Do not depend on the relative order of finalization of objects.
4. Do not use explicit layout of data.
5. Do not rely on the relative order of exceptions within a single CIL instruction or a given library method call.

Annex F Imprecise faults

This clause contains only informative text

Some CIL instructions perform implicit run-time checks that ensure memory and type safety. Originally, the CLI guaranteed that exceptions were precise, meaning that program state was preserved when an exception was thrown. However, enforcing precise exceptions for implicit checks makes some important optimizations practically impossible to apply. Programmers can now declare, via a custom attribute, that a method is "relaxed", which says that exceptions arising from implicit run-time checks need not be precise.

Relaxed checks preserve verifiability (by preserving memory and type safety) while permitting optimizations that reorder instructions. In particular, it enables the following optimizations:

- Hoisting implicit run-time checks out of loops.
- Reordering loop iterations (e.g., vectorization and automatic multithreading)
- Interchanging loops
- Inlining that makes an inlined method as least as fast as the equivalent macro

F.1 Instruction reordering

Programs that always perform explicit checks and explicit throws, instead of relying on implicit run-time checks, are never visibly affected by relaxation, except for variations already permitted by the existing CLI standard (e.g., non-determinism of cross-thread non-volatile reads and writes). Furthermore, only control dependences induced by implicit run-time checks are relaxed. Nonetheless, *data* dependences must be respected.

Authors of strict methods can reason about their behavior without knowing details about whether their callers or callees are relaxed, because strict instructions act as a fence. On the other hand, we want calls from E-relaxed methods to E-relaxed methods to be inlinable "as if" they were inlined by hand at the source level. That is why an E-relaxed sequence is allowed to span between methods.

F.2 Inlining

Inliners must be careful when dealing with a call to a method of different strictness. A call from a method to a more relaxed method can be inlined, conservatively, by treating the callee as strict as the caller; i.e., by ignoring any additional latitude granted the callee. Otherwise, if the strictness of the caller and callee differ, inlining the call requires either careful tracking of whether each check is relaxed or strict, or demoting the entire caller and inlined copy of the callee to a strictness that is at least as strict as the strictnesses of the caller and callee.

F.3 Finally handlers still guaranteed once a try block is entered

Because relaxed sequences cannot span across protected non-trivial region boundaries, this guarantee still holds. This is essential for preserving the usual idiom for acquiring and releasing a resource: [*Example*:

```
bool acquired = false;
try {
      acquire(ref acquired);
      S1;
} finally {
      if (acquired) release resource;
}
```

end example]

Quite often, the programmer knows little about how S1 might fail. If the "acquire", S1, and "release" were allowed to be part of the same relaxed sequence, and S1 failed, then the acquire and/or release portions could be suppressed at whim (by the rest of the rules). By forcing the three parts to be in three separate sequences, we eliminate problems with regard to S1 failing. Of course, we do not eliminate problems that might arise if something else in the sequence for "acquire" fails, but that is a problem that can't be dealt with at the CLI level, and must be left to the programmer.

Relaxed sequences are allowed to span trivial region boundaries because optimizers were already allowed to remove such regions even when strict exception handling is specified.

F.4 Interleaved calls

One potential hazard that users should look out for is that when a relaxed method calls another relaxed method, checks can appear to migrate from callee to caller and vice versa. Thus, methods that enforce program invariants that must be maintained in spite of faults should be marked as being strict for faults whose retiming may break the invariant.

For example, the method `T.M` below keeps `x+y` invariant.

[*Example*:

```
.class M {
  .field public int32 x;
  .field public int32 y;

  .method public void T() cil managed {
        .maxstack 2
        ldarg.0                 // Compute x=x-1

        dup
        ldfld x
        ldc.i4.1
        sub
        stfld x

        ldarg.0                 // Compute y=y+1

        dup
        ldfld y
        ldc.i4.1
        add
        stfld y
  }
  ...
}
```

end example]

If this method is relaxed, and the caller is also relaxed, then the caller would be allowed, in the absence of constraining data or control dependences, to interleave the call with other instructions in the caller. If one of those other interleaved instructions faults, then any or all of `M`'s side effects might be suppressed. Thus, method `M` should be marked as strict if it is important to prevent a fault from destroying the invariant.

This "interference" from the caller is potentially annoying, but seems to be intrinsic to any definition of relaxed exceptions that permits both:

1. instruction reordering and
2. inlined method calls are at least as fast as manual inlining.

F.4.1 Rejected notions for fencing

This subclause explains why some alternative ideas for "check fence" rules that were rejected.

Volatile operations were a candidate, since they already prevent some kinds of reordering. Treating volatile memory operations as check fences would prevent interference in critical sections. However, there are two arguments against this. First, not all situations that need check fences have anything to do with volatile operations. Second, it would penalize volatile references, which exist for sake of fast cross-thread communication.

F.5 Examples

This subclause shows some classic optimizations, and how relaxed exceptions make them much easier to apply than strict exceptions.

F.5.1 Hoisting checks out of a loop

In a relaxed method, bounds checks for arithmetically progressing indices can be hoisted out of a loop, and only the extremes are checked. For example, consider:

```
for( int i=lower; i<upper; ++i ) {
  a[i] = b[i];
  c[i] = d[i];
}
```

In a strict method, the bounds checks on `a` and `b` are difficult to hoist, because the assignment to `c[i]` is control-dependent on success of all the bounds checks in the loop. If a fault causes the loop to end prematurely, the initial prefixes of `a` and `c` must be written up to where the fault occurred. The hoisting can be of course done via “two versioning”, but that would double the size of the generated code.

In relaxed methods, the bounds checks can easily be hoisted without resorting to two-versioning, so that the code executes as if written:

```
if(lower < upper) {
  // "Landing pad" in compiler parlance.
  if( lower < 0 || upper < a.Length || upper < b.Length || upper
< c.Length
      || upper < d.Length)
        throw IndexOutOfRangeException;

  int i=lower;
  do {
        a[i] = b[i];       // Unchecked
        c[i] = d[i];       // Unchecked
  } while( ++i<upper );
}
```

Notice that the rewrite implicitly hoists the check for `NullReferenceException` too. With strict exceptions, that hoisting would not be valid, because perhaps `a[0]=b[0]` succeeds but then `c` is null. For similar reasons, relaxed exceptions (specifically, with the exceptions indicated by `CompilationRelaxations.RelaxedArrayExceptions` and `CompilationRelaxations.RelaxedNullReferenceException` relaxed) enables the hoisting of the checks for `ArrayTypeMismatchException` for both assignments. Notice that relaxation allows the checks to be hoisted, not removed.

F.5.2 Vectorizing a loop

Vectorizing a loop usually requires knowing two things:

1. The loop iterations are independent
2. The number of loop iterations is known.

In a method relaxed for the checks that might fault, part 1 is frequently false, because the possibility of a fault induces a control dependence from each loop iteration to succeeding loop iterations. In a relaxed method, those control dependences can be ignored.

In most cases, relaxed methods simplify vectorization by allowing checks to be hoisted out of a loop. Nevertheless, even when such hoisting is not possible, ignoring cross-iteration dependences implied by faults can be crucial to vectorization for “short vector” SIMD hardware such as IA-32 SSE or PowerPC Altivec. For example, consider this loop:

```
for (k = 0; k < n; k++) {
  x[k] = x[k] + y[k] * s[k].a;
}
```

where `s` is an array of references. The checks for null references cannot be hoisted out of the loop, even in a relaxed context. But relaxed does allow “unroll-and-jam” to be applied successfully. The loop can be unrolled by a factor of 4 to create aggregate iterations, and the checks hoisted to the top of each aggregate iteration.

F.5.3 Autothreading a loop

Below is a C# rendition of the key routine for a sparse matrix multiply from the SciMark 2.0 suite:

```
int M = row.Length - 1;

for (int r=0; r<M; r++) {
  double sum = 0.0;
  int rowR = row[r];
  int rowRp1 = row[r + 1];
  for (int i = rowR; i < rowRp1; i++)
         sum += x[ col[i] ] * val[i];
  y[r] = sum;
}
```

This is an attractive candidate for parallelizing the outer loop. In a strict method, doing so is quite difficult; either we have to know `x[col[i]]` never faults, or have a way to make the writes to `y[r]` speculative.

If the method is relaxed for the possible faults, parallelizing the outer loop is only a matter of solving the usual data dependence problem (“Does `y[r]` ever alias `x[col[i]]`”). If any iteration of the loop faults, the relaxed exceptions allows the other iterations to quit early or keep going without concern for what state they leave `y` in.

Annex G Parallel library

This clause contains only informative text

This Annex shows several complete examples written using the parallel library

The classes in `System.Threading.Parallel` enable you to write parallel loops that take advantage of hardware that can execute multiple threads in parallel, without having to get involved in the details of dispatching and synchronizing multiple threads. [*Example*: The library lets you take an ordinary sequential loop like this:

```
for( int i=0; i<n; ++i ) {
    loop body
}
```

and rewrite it as a parallel loop like this:

```
new ParallelFor().Run( delegate( int i ) {
    loop body
});
```

end example]

G.1 Considerations

The programmer is responsible for ensuring that the loop iterations are *independent* (for sake of correctness) and have sufficient *grain size* (for sake of efficiency.) Loop iterations are *independent* if they can be carried out in arbitrary order, or concurrently and still produce the right answer. The *grain size* is the amount of work performed by a loop iteration. If the grain size is too small, the overhead (calling the delegate, synchronizing with other threads, etc.) may overwhelm the intended work. The ideal is to make the grain size large and uniform, but not so large as to make it difficult to distribute work evenly across physical threads.

For efficiency, `ParallelFor` is the preferred loop class when there is a choice. It tends to be the most efficient because it has the least general iteration space.

G.2 ParallelFor

`ParallelFor` should be used when parallelizing a loop whose index takes on values from 0 to *`n-1`*. Below is an example of how `ParallelFor` might be used in C# to parallelize the iterations in a cellular automaton on a grid. The variables `oldState` and `newState` are two-dimensional arrays the respectively hold the old and new states of the cells. [*Example*:

```
int n = oldState.GetLength(0);
new ParallelFor(n-2).Run(delegate(int iteration) {
    int i = iteration+1;
    for (int j = 1; j < n-1; j++){
        int count =
           (oldState[i-1,j-1] + oldState[i-1,j] + oldState[i-1,j+1] +
            oldState[  i,j-1] +                   oldState[  i,j+1] +
            oldState[i+1,j-1] + oldState[i+1,j] + oldState[i+1,j+1]);
       byte s = (count | oldState[i, j]) == 3 ? Live : Dead;
       newState[i, j] = s;
    }
});
```

end example]

There are two key points to notice. First, the outer loop logically iterates `i` from `1` to `n-1`. However, the `ParallelFor` class always iterates starting at 0. Hence the desired logical value of `i` is computed from the physical loop iteration number `iteration`. Second, outer loop is parallel; the inner loop is sequential. In general, if the loop iterations are independent for both inner and outer loops, it is better to parallelize the outer loop because doing so yields the largest grain size.

G.3 ParallelForEach

`ParallelForEach` should be used to parallelize a loop that iterates over a collection that supports the usual enumerator pattern. Below is an example that iterates over a list of file names. [*Example:*

```
List<string> files = ...;
new ParallelForEach(files).Run( delegate(filename) {
    FileStream f = new FileStream( filename, FileMode.Open );
    ...read file f and process it...
    f.Close();
});
```

end example]

G.4 ParallelWhile

Use `ParallelWhile` to parallelize a loop over a collection that grows while it is being processed. Below is an excerpt showing how `ParallelWhile` might be used for parallel update of cells in a spreadsheet. Each cell is presumed to have a set `Successors` of cells that depend upon it, and a field `PredecessorCount` that is initially zero. Each cell must be updated before any of its successors is updated.

[*Example:*

```
void UpdateAll() {
    // Phase 1: Count predecessors
    foreach (Cell c in SetOfAllCells)
        foreach (Cell dependent in currentCell.Sucessors)
            ++dependent.PredecessorCount

    // Phase 2: Find cells with no predecessors
    ParallelWhile<Cell> parallelWhile = new ParallelWhile<Cell>();
    foreach (Cell c in SetOfAllCells)
        if (c.PredecessorCount]==0)
            parallelWhile.Add(c);

    // Phase 3: Do the updating
    parallelWhile.Run( delegate(Cell c) {
        ....update value of cell c...
       foreach (Cell dependent in c.Sucessors)
           if (Interlocked.Decrement(ref
dependent.PredecessorCount)==0)
               parallelWhile.Add(dependent);
    });
}
```

end example]

The example is structured as a classic topological sort. Phases 1 and 2 are sequential code. Because they are sequential, they do not have to update `PredecessorCount` in a thread-safe manner. Phase 3 is parallel: it starts processing all cells that phase 2 found were ready to update, and any cells found by phase 3 itself that were found ready to run. Because phase 3 is parallel, it updates `PredecessorCount` in a thread-safe manner.

G.5 Debugging

During initial debugging, set `System.Threading.Parallel.ParallelEnvironment.MaxThreads` to 1, which causes sequential execution of the parallel loop classes. Once your code runs correctly sequentially, experiment with setting `System.Threading.Parallel.ParallelEnvironment.MaxThreads` to higher values. In final production code, it is preferable to not set it at all, because it affects parallel loops everywhere in the application domain.