L2/02-039

ECMA/TC39/01/19A

Common Language Infrastructure (CLI)

Partition V:
Annexes

Annex A Scope

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 may 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.

1

Annex B Sample Programs

This chapter 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.

```
.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’
```

B.2. Using Value Types

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

```
.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
  }
```

1

Annex C CIL Assembler Implementation

This chapter contains only informative text

This section 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, Section C.1 and Section 0 may prove 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 Section 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
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	dup	implicitcom	ldc.i4.4
conv.ovf.i4	endfault	implicitres	ldc.i4.5
conv.ovf.i4.un	endfilter	import	ldc.i4.6
conv.ovf.i8	endfinally	in	ldc.i4.7
conv.ovf.i8.un	endmac	inheritcheck	ldc.i4.8
conv.ovf.u	enum	init	ldc.i4.M1
conv.ovf.u.un	error	initblk	ldc.i4.m1
conv.ovf.u1	explicit	initobj	ldc.i4.s
conv.ovf.u1.un	extends	initonly	ldc.i8
conv.ovf.u2	extern	instance	ldc.r4
conv.ovf.u2.un	false	int	ldc.r8
conv.ovf.u4	famandassem	int16	ldelem.i
conv.ovf.u4.un	family	int32	ldelem.i1
conv.ovf.u8	famorassem	int64	ldelem.i2
conv.ovf.u8.un	fastcall	int8	ldelem.i4
conv.r.un	fastcall	interface	ldelem.i8
conv.r4	fault	internalcall	ldelem.r4
conv.r8	field	isinst	ldelem.r8
conv.u	filetime	iunknown	ldelem.ref
conv.u1	filter	jmp	ldelem.u1
conv.u2	final	lasterr	ldelem.u2
conv.u4	finally	lcid	ldelem.u4
conv.u8	fixed	ldarg	ldelem.u8
cpblk	float	ldarg.0	ldelema
cpobj	float32	ldarg.1	ldfld
currency	float64	ldarg.2	ldflda
custom	forwardref	ldarg.3	ldftn
date	fromunmanaged	ldarg.s	ldind.i
decimal	handler	ldarga	ldind.i1
default	hidebysig	ldarga.s	ldind.i2
default	hresult	ldc.i4	ldind.i4
demand	idispatch	ldc.i4.0	ldind.i8
deny	il	ldc.i4.1	ldind.r4
div	illegal	ldc.i4.2	ldind.r8
div.un	implements	ldc.i4.3	ldind.ref

ldind.u1	modreq	pop	sequential
ldind.u2	mul	prefix1	serializable
ldind.u4	mul.ovf	prefix2	shl
ldind.u8	mul.ovf.un	prefix3	shr
ldlen	native	prefix4	shr.un
ldloc	neg	prefix5	sizeof
ldloc.0	nested	prefix6	special
ldloc.1	newarr	prefix7	specialname
ldloc.2	newobj	prefixref	starg
ldloc.3	newslot	prejitdeny	starg.s
ldloc.s	noappdomain	prejitgrant	static
ldloca	noinlining	preservesig	stdcall
ldloca.s	nomachine	private	stdcall
ldnull	nomangle	privatescope	stelem.i
ldobj	nometadata	protected	stelem.i1
ldsfld	noncasdemand	public	stelem.i2
ldsflda	noncasinheritance	readonly	stelem.i4
ldstr	noncaslinkdemand	record	stelem.i8
ldtoken	nop	refany	stelem.r4
ldvirtftn	noprocess	refanytype	stelem.r8
leave	not	refanyval	stelem.ref
leave.s	not_in_gc_heap	rem	stfld
linkcheck	notremotable	rem.un	stind.i
literal	notserialized	reqmin	stind.i1
localloc	null	reqopt	stind.i2
lpstr	nullref	reqrefuse	stind.i4
lpstruct	object	reqsecobj	stind.i8
lptstr	objectref	request	stind.r4
lpvoid	opt	ret	stind.r8
lpwstr	optil	rethrow	stind.ref
managed	or	retval	stloc
marshal	out	rtspecialname	stloc.0
method	permitonly	runtime	stloc.1
mkrefany	pinned	safearray	stloc.2
modopt	pinvokeimpl	sealed	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
unicode
unmanaged
unmanagedexp
unsigned
unused
userdefined
value
valuetype

vararg
variant
vector
virtual
void
volatile.
wchar
winapi
with
wrapper
xor

1

C.2. CIL Opcode Descriptions

This Section 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 may 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 may vary by instruction), next is an object reference
   m. PopRef+PopI+PopI - Top of stack has two integers (size may 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 two byte encoding, or 0xFF if single byte instruction.
9. One byte encoding, or second byte of two-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 singlely 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 section 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 Section C.1 with the information in the following table:

Table 1: Instruction Syntax classes

Grammar Class	Format(s) Specified in Section 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 chapter contains only informative text

This chapter describes the guidelines that were used in the design of the class libraries, including naming conventions and coding patterns. They are intended to give guidance to anyone who is extending the libraries, including:

- Implementers of the CLI who wish to extend the libraries beyond those specified in this Standard
- Implementers of libraries that will run on top of the CLI and wish their libraries to be consistent with the standard libraries
- Future standards efforts aimed at refining the existing libraries or defining additional libraries.

As with any set of guidelines, they should be applied with an eye toward the end goal of consistency but understanding that for functionality, performance, or external compatibility reasons they may require modification or simply prove inappropriate in particular cases. The guidelines should not be applied blindly, and they should be revisited periodically to ensure that they remain viable.

Throughout this chapter, we use the following convention:

- <u>Do</u> means that the described practice should be followed where possible
- <u>Do not</u> means that the described practice should be avoided where possible
- <u>Consider</u> means that the described practice is often helpful but there are common cases where it is impractical or inadvisable; thus, the practice should be carefully considered but may not be appropriate.

D.1. Naming Guidelines

One of the most important elements of predictability and discoverability in a managed class library is the use of a consistent naming pattern. Many of the most common user questions should not arise once these conventions are understood and widely used.

There are three elements of naming guidelines.

- **Case:** Use the correct capitalization style.
- **Mechanics:** Use nouns for classes, verbs for methods, etc.
- **Word Choice:** Use terms consistently across libraries.

The following section describes rules for case and mechanics, and some philosophy regarding word choice.

D.1.1. Capitalization Styles

The following section describes different ways of capitalizing identifiers. These terms will be referred to throughout the rest of this document.

D.1.1.1. Pascal Casing

This convention capitalizes the first character of each word as in the following example.

```
BackColor
```

D.1.1.2. Camel Casing

This convention capitalizes the first character of each word except the first word as in the following example. `backColor`

D.1.1.3. Upper Case

Only use all upper case letters for identifiers if it contains an abbreviation that is two characters long or less. Identifiers of three or more characters should us Pascal Casing.

```
System.IO
System.Web.UI
System.CodeDom
```

D.1.1.4. Capitalization summary

The following table describes the capitalization rules for different types of identifiers.

Type	**Case**	**Notes**
Class	PascalCase	
Enum values	PascalCase	
Enum type	PascalCase	
Events	PascalCase	
Exception class	PascalCase	Ends with the suffix Exception.
Final Static field	PascalCase	
Interface	PascalCase	Begins with the prefix **I.**
Method	PascalCase	
Namespace	PascalCase	
Property	PascalCase	
Public Instance Field	PascalCase	Rarely used, prefer properties.
Protected Instance Field	camelCase	Rarely used, prefer properties.
Parameter	camelCase	

D.1.2. Word Choice

- <u>Do</u> avoid using class names duplicated in heavily used namespaces. For example, do not use any of the following for a class name.

  ```
  System
  Collections
  Forms
  UI
  ```

- <u>Do</u> avoid using identifiers that conflict with the following keywords.

alias	and	ansi	as	assembly
auto	base	bool	boolean	byte
call	case	catch	char	class
const	current	date	decimal	declare
default	delegate	dim	do	double
each	else	elseif	end	enum
erase	error	eval	event	exit
extends	finalize	finally	float	for
friend	function	get	goto	handles
if	implements	import	imports	in
inherit	inherits	instanceof	int	integer

interface	is	let	lib	like
lock	long	loop	me	mod
module	namespace	new	next	not
nothing	null	object	on	or
overloads	override	overrides	package	private
property	protected	public	raise	readonly
redim	rem	resume	return	select
self	set	shared	short	single
static	step	stop	string	structure
sub	synchronize	synchronized	then	this
throw	to	try	typeof	unlock
until	use	uses	using	var
void	volatile	when	while	with
xor	FALSE	TRUE		

- Do not use abbreviations in identifiers (including parameter names).
- If you must use abbreviations, do use camelCasing for any abbreviation over two characters long, even if this is not the standard abbreviation.

D.1.3. Case Sensitivity

Do not use names that require case sensitivity. Components must be fully usable from both case-sensitive and case-insensitive languages. Since case-insensitive languages cannot distinguish between two names within the same context that differ only by case, components must avoid this situation.

- Do not have two namespaces whose names differ only by case

```
namespace ee.cummings;
namespace Ee.Cummings;
```

- Do not have a function with two parameters whose names differ only by case.

```
void foo(string a, string A)
```

- Do not have a namespace with two types whose names differ only by case.

```
System.Drawing.Point p;
System.Drawing.POINT pp;
```

- Do not have a type with two properties whose names differ only by case.

```
int Foo {get, set};
int FOO {get, set}
```

- Do not have a type with two methods whose names differ only by case.

```
void foo();
void Foo();
```

D.1.4. Avoiding Type Name Confusion

Different languages use different terms to identify the fundamental managed types. Designers must avoid using language-specific terminology. Follow the rules described in this section to avoid type name confusion.

- Do use semantically interesting names rather than type names.

- In the rare case that a parameter has no semantic meaning beyond its type, use a generic name. For example, a class that supports writing a variety of data types into a stream might have the following methods.

```
void Write(double value);
void Write(float value);
void Write(long value);
void Write(int value);
void Write(short value);
```

  The above example is preferred to the following language-specific alternative.

```
void Write(double doubleValue);
void Write(float floatValue);
void Write(long longValue);
void Write(int intValue);
void Write(short shortValue);
```

In the extremely rare case that it is necessary to have a uniquely-named method for each fundamental data type, <u>do</u> use the following **universal type** names.

C# type name	**ILAsm representation**	**Universal type name**
sbyte	int8	**SByte**
byte	unsigned int8	**Byte**
short	int16	**Int16**
ushort	unsigned int16	**UInt16**
int	int32	**Int32**
uint	unsigned int32	**UInt32**
long	int64	**Int64**
ulong	unsigned int64	**UInt64**
float	float32	**Single**
double	float64	**Double**
bool	int32	**Boolean**
char	unsigned int16	**Char**
string	System.String	**String**
object	System.Object	**Object**

A class that supports reading a variety of data types from a stream might have the following methods.

```
double ReadDouble();
float ReadSingle();
long ReadInt64();
int ReadInt32();
short ReadInt16();
```

The above example is preferred to the following language-specific alternative.

```
double ReadDouble();
float ReadFloat();
long ReadLong();
```

```
int ReadInt();
short ReadShort();
```

D.1.5. Namespaces

The following example illustrates the general rule for naming namespaces.

```
CompanyName.TechnologyName
```

Therefore, we should expect to see namespaces like the following.

```
Microsoft.Office
PowerSoft.PowerBuilder
```

- Do avoid the possibility of two published namespaces having the same name, by prefixing namespace names with a company name or other well-established brand. For example, `Microsoft.Office` for the Office Automation Classes provided by Microsoft.
- Do use PascalCasing, and separate logical components with periods (For example, `Microsoft.Office.PowerPoint`). If your brand employs non-traditional casing, do follow the casing defined by your brand, even if it deviates from normal namespace casing (For example, `NeXT.WebObjects`, and `ee.cummings`).
- Do use plural namespace names where appropriate. For example, use `System.Collections` not `System.Collection`. Exceptions to this rule are brand names and abbreviations. For example, use `System.IO` not `System.IOs`.
- Do not specify the same name for namespaces and classes. For example, do not use `Debug` for a namespace name and also provide a class named `Debug`.

D.1.6. Classes

- Do name classes with nouns or noun phrases.
- Do use PascalCasing.
- Do use abbreviations in class names sparingly.
- Do not use any type of class prefix (such as **C**).
- Do not use the underscore character.
- Occasionally, it is necessary to have a class name that begins with **I,** that is not an interface. This is acceptable as long as the character that follows **I** is lower case (For example, IdentityStore).

The following are examples of correctly named classes.

```
public class FileStream
{
}
public class Button
{
}
public class String
{
}
```

D.1.7. Interfaces

- Do name interfaces with nouns or noun phrases, or adjectives describing behaviour. For example, `IComponent` (descriptive noun), `ICustomAttributeProvider` (noun phrase), and `IPersistable` (adjective) are appropriate interface names.

- Do use **`PascalCasing`**.
- Do use abbreviations in interface names sparingly.
- Do not use the underscore character.
- Do prefix interface names with the letter **I**, to indicate that the type is an interface.
- Do use similar names when defining a class/interface pair where the class is a standard implementation of the interface. The names should differ only by the letter **I** prefix on the interface name.

The following example illustrates these guidelines for the interface `IComponent` and its standard implementation, the class `Component`.

```
public interface IComponent
{
}
public class Component : IComponent
{
}
public interface IServiceProvider
{
}
public interface IFormattable
{
}
```

D.1.8. Attributes

- Do add the **Attribute** suffix to custom attribute classes as in the following example.

  ```
  public class ObsoleteAttribute
  {
  }
  ```

D.1.9. Enums

- Do use PascalCasing for an `enum` type.
- Do use PascalCasing for an `enum` value name.
- Do use abbreviations in `enum` names sparingly.
- Do not use a prefix on `enum` names (For example, adXXX for ADO enums, rtfXXX for rich text enums, etc.).
- Do not use an `Enum` suffix on `enum` types.
- Do use a singular name for an `enum`.
- Do use a plural name for bit fields.

D.1.10. Fields

- Do use camelCasing (except for static fields, see clause D.1.10.1).
- Do not abbreviate field names.

  Spell out all the words used in a field name. Only use abbreviations if developers generally understand them. Do not use uppercase letters for field names. For example:

  ```
  class Foo
  {
  ```

```
   string url;
   string destinationUrl;
}
```

- Do not use Hungarian notation for field names. Good names describe semantics, not type.
- Do not use a prefix for field names.
- Do not include a prefix on a field name, for example 'g_' or 's_' to distinguish static vs. non-static fields.

D.1.10.1. Static Fields

- Do name static fields with nouns, noun phrases, or abbreviations for nouns.
- Do not use a prefix for static field names.
- Do name static fields with PascalCasing.
- Do not prefix static field names with Hungarian type notation.

D.1.11. Parameter Names

- Do use descriptive parameter names. Parameter names should be descriptive enough that in most scenarios the name of the parameter and its type can be used to determine its meaning.
- Do name parameters with camelCasing.
- Do use names based on a parameter’s meaning rather than names based on the parameter’s type. We expect development tools to provide the information about type in a useful manner, so the parameter name can be put to better use describing semantics rather than type. Occasional use of type-based parameter names is entirely appropriate.
- Do not use **reserved** parameters. If more data is needed in the next version, a new overload can be added.
- Do not prefix parameter names with Hungarian type notation.

```
Type GetType (string typeName)
string Format (string format, object [] args)
```

D.1.12. Method Names

- Do name methods with PascalCasing as in the following examples.

```
RemoveAll()
GetCharArray()
Invoke()
```

- Do not use Hungarian notation.
- Do name methods with verbs or verb phrases.

D.1.13. Property Names

- Do name properties using a noun or noun phrase.
- Do name properties with PascalCasing.
- Do not use Hungarian notation.

D.1.14. Event Names

- Do name events using PascalCasing.

- Do not use Hungarian notation.
- Do name event handlers (delegate types) with the **EventHandler** suffix as in the following example.

```
public delegate void MouseEventHandler(object sender, MouseEvent e);
```

- Consider using two parameters named **sender** and **e**.

  The sender parameter represents the object that raised the event. The sender parameter is always of type **object**, even if it is possible to employ a more specific type.

  The state associated with the event is encapsulated in an instance of an event class named **e**. Use an appropriate and specific event class for its type.

```
public delegate void MouseEventHandler(object sender, MouseEvent e);
```

- Do name event argument classes with the **EventArgs** suffix as in the following example.

```
public class MouseEventArgs : EventArgs
{
   int x;
   int y;
   public MouseEventArgs(int x, int y)
       { this.x = x; this.y = y; }
   public int X { get { return x; } }
   public int Y { get { return y; } }
}
```

- Do name event names that have a concept of pre and post using the present and past tense (do not use the BeforeXxx\AfterXxx pattern). For example, a close event that can be canceled would have a Closing and Closed event.
- Consider naming events with a verb.

D.2. Type Member Usage Guidelines

D.2.1. Property Usage Guidelines

- Do see clause D.2.1.1 on choosing between properties and methods.
- Do not use properties and types with the same name.

  Defining a property with the same name as a type can cause ambiguity in some programming languages. It is best to avoid this ambiguity unless there is a clear justification for not doing so.
- Do preserve the previous value if a property set throws an exception.
- Do allow properties to be set in any order. Properties should be stateless with respect to other properties.

  It is often the case that a particular feature of an object will not take effect until the developer specifies a particular set of properties, or until an object has a particular state. Until the object is in the correct state, the feature is not active. When the object is in the correct state, the feature automatically activates itself without requiring an explicit call. The semantics are the same regardless of the order in which the developer sets the property values or how the developer gets the object into the active state.

D.2.1.1. Properties vs. Methods

Library designers sometimes face a decision between a property and a method. Use the following guidelines to help you choose between these options. The philosophy here is that users will think of properties as though they were fields, hence methods are preferred where the intuitive semantics or performance differ from those of fields.

- Do use a property if the member has a logical backing store.
- Do use a method in the following situations.
  - The operation is a conversion (such as `Object.ToString()`)
  - The operation is expensive (orders of magnitude slower than a field set would be).
  - Obtaining a property value using the `Get` accessor has an observable side effect.
  - Calling the member twice in succession results in different results.
  - The order of execution relative to other properties is important.
  - The member is static but returns a mutable value.
  - The member returns an array.

    Properties that return arrays can be very misleading. Usually it is necessary to return a copy of the internal array so that the user cannot change internal state. This, coupled with the fact that a user could easily assume it is an indexed property, leads to inefficient code. In the following example, each call to the Methods property creates a copy of the array. That would be 2n+1 copies for this loop.

```
Type type = //get a type somehow
for (int i = 0; i < type.Methods.Length; i++)
{
   if (type.Methods[i].Name.Equals ("foo"))
   {...}
}
```

D.2.1.2. Read-Only and Write-Only Properties

- Do use Read-only properties when the user cannot change the logical backing data field.
- Do not use Write-only properties.

D.2.1.3. Indexed Property Usage

- Do use only one indexed property per class and make it the default indexed property for that class.
- Do not use non-default indexed properties.
- Do use the name **`Item`** for indexed properties unless there is an obviously better name (for example, a **`Chars`** property on **`string`** is better than **`Item`**).
- Do use indexed properties when the logical backing store is an array.
- Do not provide both indexed properties and methods that are semantically equivalent to two or more overloaded methods.

```
MethodInfo Type.Method[string name]       ;; Should be method
MethodInfo Type.GetMethod (string name, boolean ignoreCase)
```

D.2.2. Event Usage Guidelines

- Do use the "raise" terminology for events rather than "fire" or “trigger” terminology.
- Do use a return type of void for event handlers.
- Do make Event classes extend the class `System.EventArgs`
- Do implement `AddOn<EventName>` and `RemoveOn<EventName>` for each event.

- Do use a **`family virtual`** method to raise each event.

  This is not appropriate for sealed classes, because classes cannot be derived from them. The purpose of the method is to provide a way for a derived class to handle the event using an override. This is more natural than using delegates in the case where the developer is creating a derived class.

  The derived class can choose not to call the base during the processing of **`On<EventName>`**. Be prepared for this by not including any processing in the **`On<EventName>`** method that is required for the base class to work correctly.

- Do assume that anything can go in an event handler.

  Classes are ready for the handler of the event to do almost anything, and in all cases the object is left in a good state after the event has been raised. Consider using a try/finally block at the point where the event is raised. Since the developer can call back on the object to perform other actions, do not assume anything about the object state when control returns to the point at which the event was raised

D.2.3. Method Usage Guidelines

- Do use non-virtual methods unless overriding is intended by the design. Providing the ability to override a method (i.e. making the method virtual) implies that the design of the type is independent of details of the method’s implementation; this is rarely true without careful design of the type.

- Do use method overloading when you provide different methods that do semantically the same thing.

- Do favor method overloading to default arguments. Default arguments are not allowed in the common language specification (CLS).

```
int String.IndexOf (String name);

int String.IndexOf (String name, int startIndex);
```

- Do use default values correctly.
  In a family of overloaded methods the complex method should use parameter names that indicate a change from the default state assumed in the simple method.
  For example, in the code below, the first method assumes the look-up will not be case sensitive. In method two we use the name *ignoreCase* rather than *caseSensitive* because the former indicates how we are changing the default behavior.

```
MethodInfo Type.GetMethod(String name);  //ignoreCase = false

MethodInfo Type.GetMethod (String name, boolean ignoreCase);
```

  It is very common to use a zero’ed state for the default value (such as: 0, 0.0, false, “”, etc).

- Do be consistent in the ordering and naming of method parameters.

  It is common to have a set of overloaded methods with an increasing number of parameters to allow the developer to specify a desired level of information. The more parameters specified, the more detail that is specified. All the related methods have a consistent parameter order and naming pattern. Each of the method variations has the same semantics for their shared set of parameters.

  This consistency is useful even if the parameters have different types.

  The only method in such a group that should be virtual is the one that has the most parameters.

- Do use method overloading for variable numbers of parameters.

  Where it is appropriate to have variable numbers of parameters to a method, use the convention of declaring N methods with increasing numbers of parameters, and also provide a method which takes an array of values for numbers greater than N. N=3 or N=4 is appropriate for most cases. Only the method that takes the array should be virtual.

- Do make only the most complete overload virtual (if extensibility is needed) and define the other operations in terms of it.

```
public int IndexOf (string s)
{ return IndexOf (s, 0); }
public int IndexOf (string s, int start)
{ return IndexOf (s, startIndex, s.Length); }
public virtual int IndexOf (string s, int start, int count)
{ //do real work }
```

- Do use the **`ParamsAttribute`** pattern for defining methods with a variable number of arguments.

```
void Format (string formatString, params object [] args)
```

- Consider using the varargs ("…") calling convention to provide variable number of arguments, but do not use this without providing an alternate mechanism to accomplish the same thing since it is not CLS compliant.

- Consider providing special-case code for a small number of arguments to a method that takes a variable number of arguments, but only where the performance gained is significant. When this approach is taken it becomes difficult to allow the method to be overridden because all the special cases must be overridden as well.

D.2.4. Constructor Usage Guidelines

- Do have only a default **`private`** constructor (or no constructor at all) if there are only static methods and properties on a class.

- Do minimal work in the constructor.

- Do provide a **`family`** constructor that can be used by types in a derived class.

- Do not provide an empty default constructor for value types.

- Do use parameters in constructors as shortcuts for setting properties.

  There should be no difference in semantics between using the empty constructor followed by some calls to property setters, and using a constructor with multiple arguments.

- Do be consistent in the ordering and naming of constructor parameters.

  A common pattern for constructor parameters is to provide an increasing number of parameters to allow the developer to specify a desired level of information. The more parameters that are specified, the more detail that is specified. For all of the following constructors, there is a consistent order and naming of the parameters.

D.2.5. Field Usage Guidelines

- Do not use instance fields that are **`public`** or **`family`**.

- Consider providing `get` and `set` property accessors for fields instead of making them `public`.

- Do use a `family` property that returns the value of a `private` field to expose a field to a derived class. By not exposing fields directly to the developer, the class can be versioned more easily for the following reasons:

  a. A field cannot be changed to a property while maintaining binary compatibility.

  b. The presence of executable code in `get` and `set` property accessors allows later improvements, such as demand-creation of an object upon usage of the property, or a property change notification.

- Do use **`readonly static`** fields instead of properties where the value is a global constant.

- Do not use **`literal static`** fields if the value can change between versions.
- Do use **`public static readonly`** fields for predefined object instances.

D.2.6. Parameter Usage Guidelines

- Do check arguments for validity.

  Perform argument validation for every **`public`** or **`family`** method and property **`set`** accessor, and throw meaningful exceptions to the developer. The `System.ArgumentException` exception, or one of its subclasses, is used in these cases.

  Note that the actual checking does not necessarily have to happen in the **`public`**/**`family`** method itself. It could happen at a lower level in some **`private`** routines. The main point is that the entire surface area that is exposed to developers checks for valid arguments.

  Parameter validation should occur before any side-effects are performed.

D.3. Type Usage Guidelines

D.3.1. Class Usage Guidelines

- Do favor using classes over any other type (i.e. interfaces or value types)

D.3.1.1. Base Class Usage Guidelines

Base classes are a useful way to group objects that share a common set of functionality. Base classes can provide a default set of functionality, while allowing customization though extension.

Add extensibility or polymorphism to your design only if you have a clear customer scenario for it.

- Do use base classes rather than interfaces.

  From a versioning perspective, interfaces are less flexible than classes. With a class, you can ship Version 1.0 and then in Version 2.0 decide to add another method. As long as the method is not abstract (that is, as long as you provide a default implementation of the method), any existing derived classes continue to function unchanged.

  Because interfaces do not support implementation inheritance, the pattern that applies to classes does not apply to interfaces. Adding a method to an interface is like adding an abstract method to a base class:any class that implements the interface will break because the class does not implement the interface's new method.

  Interfaces are appropriate in the following situations:

  - Several unrelated classes want to support the protocol.
  - These classes already have established base classes.
  - Aggregation is not appropriate or practical.

  For all other cases, class inheritance is a better model. For example, make **`IByteStream`** an interface so a class can implement multiple stream types. Make **`ValueEditor`** an abstract class because classes derived from **`ValueEditor`** have no other purpose than to edit values.

- Do provide customization through `family` methods.

  The public interface of a base class should provide a rich set of functionality for the consumer of that class. However, customizers of that class often want to implement the fewest methods possible to provide that rich set of functionality to the consumer. To meet this goal, provide a set of non-virtual or final public methods that call through to a single family method with the **`Impl`** suffix that provides implementations for such a method. This pattern is also known as the “Template Method”.

```
Public Control
{ public void SetBounds(int x, int y, int width, int height)
```

```
{ . . .
  SetBoundsImpl (…);
 }

 public void SetBounds(int x, int y,
                       int width, int height,
                       BoundsSpecified specified)
 { . . .
   SetBoundsImpl (…);
 }

 protected virtual void SetBoundsImpl
                        (int x, int y,
                         int width, int height,
                         BoundsSpecified specified)
  { // Do the real work here.
    }
 }
```

- Do define a `family` constructor on all `abstract` classes. Many compilers will insert a `public` constructor if you do not. This can be very misleading to users as it can only be called from derived classes.

D.3.1.2. Sealed Class Usage Guidelines

- Do use **`sealed`** classes if creating derived classes will not be required.
- Do use **`sealed`** classes if there are only **`static`** methods and properties on a class.

D.3.2. Value Type Usage Guidelines

- Do use a value type for types that meet all of the following criteria.
  - Act like built-in types.
  - Have an instance size under 16 bytes.
  - Value semantics are desirable.
- Do not provide a default constructor.
- Do program assuming a state where all instance data is set to zero, false, or null (as appropriate) is valid, since this will be the state if no constructor is run and there is no guarantee that a constructor will be run (unlike for classes).

D.3.2.1. Enum Usage Guidelines

- Do use an `Enum` to strongly type parameters, property and return type. This allows development tools to know the possible values for a property or parameter.
- Do use the `System.Flags` custom attribute for an **`enum`** if a bitwise OR operation is to be performed on the numeric values.
- Do use **`int32`** as the underlying type of an **`enum`**.
  An exception to this rule is if the **`enum`** represents flags and there are many flags (>32) or the **`enum`** may grow to many flags in the future or the type needs to be different than type **`int32`** for backwards compatibility.
- Do use an **`enum`** with flags attribute only if the value can be completely expressed as a set of bitflags. Do not use an **`enum`** for open sets (eg., a version number).
- Do not assume **`enum`** arguments will be in the defined range Do argument validation.
- Do favor using an **`enum`** over static final constants.

- Do use **int32** as the underlying type of an **enum** unless either of the following is true.
    a. The **enum** represents flags, and there are currently many flags (>32), or the **enum** may grow to many flags in the future.
    b. The type needs to be different than **int** for backward compatibility.
- Do not use a non-integral **enum** type. Only use **int8**, **int16**, **int32**, or **int64**.
- Do not define methods, properties or events on an **enum**.
- Do not use any suffix on **enum** types.

D.3.3. Interface Usage Guidelines

See introductory paragraph of clause D.3.1.

- Do use a class or abstract class in preference to an interface, where possible.
- Do use interfaces to provide extensibility and the ability to customize.
- Do provide a default implementation of an interface where it is appropriate. For example, `System.Collections.DictionaryBase` is the default implementation of the `System.Collections.IDictionary` interface.
- Do see clause D.3.1.1 on the versioning issues with interfaces and abstract classes.
- Do not use interfaces as empty markers. Use Custom Attributes instead.

  If you need to mark a class as having a specific attribute (such as immutable or serializable) use a custom attribute rather than an interface.
- Do implement interfaces using "method impls" (see Partition II) and **private virtual** methods if you only want the interface methods available when cast to that interface. This is particularly useful when a class or value type implements an internal interface that is of no interest to a consumer of the class or value type.

D.3.4. Delegate Usage Guidelines

Delegates are a powerful tool that allow the managed code object model designer to encapsulate method calls. They are used in two basic areas.

Event notifications

See clause D.2.2 on event usage guidelines.

Callbacks

Passed to a method so that user code can be called multiple times during execution to provide customization. The classic example of this is passing a Compare callback to a sort routine. These methods should use the Callback conventions

- Do use an Event design pattern for events (even if it is not user interface related).

D.3.5. Attribute Classes

The CLI enables developers to invent new kinds of declarative information, to specify declarative information for various program entities, and to retrieve attribute information in a runtime environment. New kinds of declarative information are defined through the declaration of attribute classes, which may have positional and named parameters.

- Do specify an **AttributeUsage** on your attributes to define their usage precisely.
- Do seal attribute classes if possible.
- Do provide a single constructor for the attribute.
- Do use a parameter to the attribute's constructor when the value of that parameter is always required to make the attribute.

- Do use a field on an attribute when the value of that property can be optionally specified to make the attribute.
- Do not name a parameter to the constructor with the same name as a field or property of the attribute.
- Do provide a read-only property with the same name (different casing) as each parameter to the constructor.
- Do provide a read-write property with the same name (different casing) as each field of the attribute.

D.3.6. Nested Types

A nested type is a type defined within the scope of another type. They are very useful for encapsulating implementation details of a type, such as an enumerator over a collection, because they can have access to private state. Public nested types are rarely used.

Do not use public nested types unless all of the following are true.

- The nested type logically belongs to the containing type.
- The nested type is not used very often, or at least not directly.

D.4. Error Raising and Handling

- Do end Exception class names with the *Exception* suffix.
- Do use these common constructors.

```
public class XxxException : Exception
{
   XxxException() { }
   XxxException(string message) { }
   XxxException(string message, Exception inner) { }
}
```

- Do use the predefined exception types. Only define new exception types for programmatic scenarios, meaning you expect users of you library to catch exceptions of this new type and perform a programmatic action based on the exception type..
- Do not derive new exceptions directly from the base class **`Exception`**. Use one of its predefined subclasses instead.
- Do use a localized description string. When the user sees an error message, it will be derived from the description string of the exception that was thrown, and never from the exception class. Include a description string in every exception.
- Do use grammatically correct error messages including ending punctuation.

  Each sentence in a description string of an exception should end in a period. This way code that generically displays an exception message to the user does not have to handle the case where a developer forgot the final period, which is relatively cumbersome and expensive.
- Do provide exception properties for programmatic access. Include extra information (besides the description string) in an exception only when there is a programmatic scenario where that additional information is useful.
- Do throw exceptions only in exceptional cases.
  - Do not use exceptions for normal or expected errors.
  - Do not use exceptions for normal flow of control.
- Do return null for extremely common error cases. For example, `File.Open` returns a null if the file is not found, but throws an exception if the file is locked.

- Do design classes such that in the normal course of use there will never be an exception thrown. For example, a `FileStream` class might expose a way of determining if the end of the file has been reached to avoid the exception that will be thrown if the developer reads past the end of the file.
- Do throw an `InvalidOperationException` if in an inappropriate state.

  The `System.InvalidOperationException` exception should be thrown if the property set or method call is not appropriate given the object's current state.
- Do throw an `ArgumentException` or create an exception derived from this class if bad parameters are passed or detected.
- Do realize that the stack trace starts at the point where an exception is thrown, not where it is created with the **`new`** operator. You should consider this when deciding where to throw an exception.
- Do throw Exceptions rather than return an error code.
- Do throw the most specific exception possible.
- Do set all the fields on the exception you use.
- Do use Inner exceptions (chained exceptions).
- Do cleanup side effects when throwing an exception. Clearly document cases where an exception may occur after a side-effect has already taken place and cannot be retracted.
- Do not assume that side-effects do not occur before an exception is thrown, but rather that the state is restored if one is thrown. That is, another thread may see the side-effect, but will then see an addition one to restore the state.

D.4.1. Standard Exception Types

The following table breaks down the standard exceptions and the conditions for which you should create a derived class.

Exception Type	Base Type	Description	Example
Exception	Object	Base class for all Exceptions.	None (use a derived class of this exception).
SystemException	Exception	Base class for all runtime generated errors.	None (use a derived class of this exception).
IndexOutOfRangeException	SystemException	Thrown only by the runtime when an array is indexed improperly.	Indexing an array outside of its valid range: arr[arr.Length+1]
NullReferenceException	SystemException	Thrown only by the runtime when a null object is referenced.	object o = null; o.ToString();
InvalidOperationException	SystemException	Thrown by methods when in an invalid state.	Calling Enumerator.GetNext() after removing an Item from the underlying collection.
ArgumentException	SystemException	Base class for all	None (use a

		Argument Exceptions. Derived classes of this exception should be thrown where applicable.	derived class of this exception).
ArgumentNullException	ArgumentException	Thrown by methods that do not allow an argument to be null.	String s = null; ”foo”.IndexOf (s);
ArgumentOutOfRangeException	ArgumentException	Thrown by methods that verify that arguments are in a given range.	String s = “string”; s.Chars[9];
InteropException	SystemException	Base class for exceptions that occur or are targeted at environments outside of the runtime.	None (use a derived class of this exception).

D.5. Array Usage Guidelines

- Do use a collection when **`Add`**, **`Remove`** or other methods for manipulating the collection are supported. This scopes all related methods to the collection.
- Do use collections to add read-only wrappers around internal arrays.
- Do use collections to avoid the inefficiencies in the following code.

```
for (int i = 0; i < obj.myObj.Count; i++)
      DoSomething(obj.myObj[i])
```

  Also see clause D.2.1.1.

- Do return an Empty array instead of a null.

  Users assume that the following code will work:

```
public void DoSomething(…)
{ int a[] = SomeOtherFunc();
  if (a.Length > 0)            // Don’t expect NULL here!
  { // do something
  }
}
```

D.6. Operator Overloading Usage Guidelines

- Do define operators on Value types that are logically a built-in language type.
- Do provide operator-overloading methods only involving the class in which the methods are defined.
- Do use the names and signature conventions described in the common language specification.
- Do not be cute.

  Operator overloading is useful in cases where it is immediately obvious what the result of the operation will be. For example, it makes sense to be able to subtract one **`Time`** value from another **`Time`** value and get a **`TimeSpan`**. However, it is not appropriate to use **`shift`** to write to a stream.

- Do overload operators in a symmetric fashion. For example, if you overload the **`Equal`** operator (**`==`**), you should also overload not equals (**`!=`**) operator.
- Do provide alternate signatures.

Most languages do not support operator overloading. For this reason it is a CLS requirement that you include a method with an appropriate domain-specific name that has the equivalent functionality as in the following example.

```
class Time {
   TimeSpan operator -(Time t1, Time t2) { }
   TimeSpan Difference(Time t1, Time t2) { }
}
```

See Partition I (Operator Overloading)

D.6.1. Implementing Equals and Operator==

Do see the section on implementing the **`Equals`** method in Section D.7.

Do implement `GetHashCode()` whenever you implement `Equals()`. This keeps `Equals()` and `GetHashCode()` synchronized.

Do override `Equals` whenever you implement `operator==` and make them do the same thing. This allows infrastructure code such as `Hashtable` and `ArrayList` which use `Equals()` to behave the same way as user code written using `operator==`.

Do override `Equals` anytime you implement `IComparable`.

Consider implementing operator overloading for ==, `!=` , <, and > when you implement `IComparable`.

Do not throw exceptions from `Equals()`, `GetHashCode()`, or `operator==` methods.

D.6.1.1. Implementing operator== on Value Types

Do overload `operator==` anytime equality is meaningful, because in most programming languages there is no default implementation of `operator==` for value types.

Consider implementing `Equals()` on ValueTypes because the default implementation on `System.ValueType` will not perform as well as your custom implementation.

Do implement `operator==` anytime you override `Equals()`

D.6.1.2. Implementing operator== on Reference Types

Do use care when implementing `operator==` on reference types. Most languages do provide a default implementation of `operator==` for reference types, therefore overriding the default implementation should be done with care. Most reference types, even those that implement `Equals()` should not override `operator==`.

Do override `operator==` if your type has value semantics (that is, if it looks like a base type such as a Point, String, BigNumber, etc.). Anytime you are tempted to overload + and - you also should consider overloading `operator==`.

D.6.2. Cast Operations (op_Explicit and op_Implicit)

- Do not lose precision in implicit casts.

  For example, there should not be an implicit cast from **`Double`** to **`Int32`**, but there may be one from **`Int32`** to **`Int64`**.

- Do not throw exceptions from implicit casts because it is very difficult for the developer to understand what is happening.

- Do provide cast operations that operate on the whole object. The value that is cast represents the whole value being cast, not one sub part. For example, it is not appropriate for a Button to cast to a string by returning its caption.

- Do not generate a semantically different value.

For example, it is appropriate to convert a **`Time`** or **`TimeSpan`** into an **`Int`**. The **`Int`** still represents the time or duration. It does not make sense to convert a file name string such as, "c:\mybitmap.gif" into a **Bitmap** object.

- Do not provide cast operations for values between different semantic domains. For example, it makes sense that an **`Int32`** can cast to a **`Double`**. It does not make sense for an **`Int`** to cast to a **`String`**, because they are in different domains.

D.7. Equals

Do see clause D.6.1 on implementing operator==.

Do override `GetHashCode()` in order for the type to behave correctly in a hashtable.

Do not throw an exception in your `Equals` implementation. Return false for a null argument, etc.

Do follow the contract defined on `Object.Equals`.

- x.Equals(x) returns true.
- x.Equals(y) returns the same value as y.Equals(x).
- (x.Equals(y) && y.Equals(z)) returns true if and only if x.Equals(z) returns true.
- Successive invocations of x.Equals(y) return the same value as long as the objects referenced by x and y are not modified.
- x.Equals(null) returns false.

For some kinds of objects, it is desirable to have `Equals` test for *value equality* instead of referential equality. Such implementations of `Equals` return true if the two objects have the same value, even if they are not the same instance. The definition of what constitutes an object's value is up to the implementer of the type, but it is typically some or all of the data stored in the instance variables of the object. For example, the value of a string is based on the characters of the string; the `Equals` method of the `String` class returns true for any two string instances that contain exactly the same characters in the same order.

When the `Equals` method of a base class provides value equality, an override of `Equals` in a class derived from that base class should invoke the inherited implementation of `Equals`.

If you choose to overload the equality operator for a given type, that type should override the `Equals` method. Such implementations of the `Equals` method should return the same results as the equality operator. Following this guideline will help ensure that class library code using `Equals` (such as `ArrayList` and `Hashtable`) behaves in a manner that is consistent with the way the equality operator is used by application code.

If you are implementing a value type, you should follow these guidelines.

- Consider overriding `Equals` to gain increased performance over that provided by the default implementation of `Equals` on `System.ValueType`.
- If you override `Equals` and the language supports operator overloading, you should overload the equality operator for your value type.

If you are implementing reference types, you should follow these guidelines.

- Consider overriding `Equals` on a reference type if the semantics of the type are based on the fact that the type represents some value(s). For example, reference types such as Point and BigNumber should override `Equals`.
- Most reference types should not overload the equality operator, even if they override `Equals`. However, if you are implementing a reference type that is intended to have value semantics, such as a complex number type, you should override the equality operator.

If you implement `IComparable` on a given type, you should override `Equals` on that type.

D.8. Callbacks

Delegates, Interfaces and Events can each be used to provide callback functionality. Each has its own specific usage characteristics that make it better suited to particular situations.

Use Events if the following are true.

- One signs up for the callback up front (typically through separate `Add` and `Remove` methods).
- Typically more than one object will care.

Use a Delegate if the following are true.

- You want a C style function pointer.
- Single callback.
- Registered in the call or at construction time (not through separate Add method)

Use an Interface if the following is true.

- The callback entails complex behavior.

D.9. Security in Class Libraries

Class library authors need to consider two perspectives with respect to security.Whether these perspectives are applicable will depend upon the class itself. Some classes, such as `System.IO.FileStream` represent objects that need protection with permissions; the implementation of these classes is responsible for checking the appropriate permissions of the caller required for each action and only allowing authorized callers to perform the actions for which they have permission. The `System.Security` namespace contains some classes to help make these checks easier. Additionally, class library code often is fully-trusted or at least highly-trusted code. Any flaws in the code represent a serious threat to the integrity of the entire security system. Therefore, extra care is required when writing class library code as detailed below.

- <u>Do</u> access protected resources only after checking the permissions of your callers, either through a declarative security attribute or an explicit call to `Demand` on an appropriate security permission object.
- <u>Do</u> assert a permission only when necessary, and always precede it by the necessary checks.
- <u>Do not</u> assume that code will only be called by callers with certain permissions.
- <u>Do not</u> define non-type-safe interfaces that might be used to bypass security.
- <u>Do not</u> expose functionality that allows a semi-trusted caller to take advantage of higher trust of the class.

D.10. Threading Design Guidelines

- <u>Do not</u> provide static methods that mutate static state.

  In common server scenarios, static state is shared across requests, which means multiple threads can execute that code at the same time. This opens up the possibility for threading bugs. <u>Consider</u> using a design pattern that encapsulates data into instances that are not shared.

- <u>Do not</u> normally provide thread safe instance state.

  By default, the library is not thread safe. Adding locks to create thread safe code decreases performance and increases lock contention (as well as opening up deadlock bugs). In common application models, only one thread at a time executes user code, which minimizes the need for thread safety. In cases where it is interesting to provide a thread safe version a `GetSynchronized()` method can be used to return a thread safe instance of that type. (See `System.Collections` for examples).

- Do make all static state thread safe.

  If you must use static state, make it thread safe. In common server scenarios, static data is shared across requests, which means multiple threads can execute that code at the same time. For this reason it is necessary to protect static state.

- Do be aware of non-atomic operations.

  Value types whose underlying representations are greater than 32 bits may have non-atomic operations. Specifically, because value types are copied bitwise (by value as opposed to by reference), race conditions can occur in what appears to be straightforward assignments within code.

  For example, consider the following code (executing on two separate threads) where the variable `x` has been declared as type `Int64`.

```
// Code executing on Thread "A".
x = 54343343433;

// Code executing on Thread "B".
x = 934343434343;
```

  At first glance it seems to indicate that there is no possibility of race conditions (since each line looks like a straight assignment operation). However, because the underlying variable is a 64-bit value type, the actual code is not doing an atomic assignment operation. Instead, it is doing a bitwise copy of two 32 bit halves. In the event of a context switch, halfway during the value type assignment operation on one of the threads, the resulting `x` variable can have corrupt data (for example, the resulting value will be composed of 32 bits of the first number, and 32 bits of the second number).

- Do be aware of method calls in locked sections.

  Deadlocks can result when a static method in class A calls static methods in class B and vice versa. If A and B both synchronize their static methods, this will cause a deadlock. You might only discover this deadlock under heavy threading stress.

  Performance issues can result when a static method in class A calls a static method in class A. If these methods are not factored correctly, performance will suffer because there will be a large amount of redundant synchronization. Excessive use of fine-grained synchronization might negatively impact performance. In addition, it might have a significant negative impact on scalability.

- Do be aware of issues with the **`lock`** statement and consider using `System.Threading.Interlocked` instead.

  It's tempting to use the **`lock`** statement in C# to solve all threading problems. But the `System.Threading.Interlocked` class is superior for updates that must be made automically

- Do avoid the need for synchronization if possible.

  Obviously for high traffic pathways it is nice to avoid synchronization. Sometimes the algorithm can be adjusted to tolerate races rather than eliminating them.

1

Annex E Portability Considerations

This chapter contains only informative text

This Chapter 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 chapter 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 may 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 may 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 may 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.