

---

---

**Common Language Infrastructure (CLI)  
Partitions I–VI**

---

---

**Partition I: Concepts and Architecture**

**Partition II: Metadata Definition and Semantics**

**Partition III: CIL Instruction Set**

**Partition IV: Profiles and Libraries**

**Partition V: Debug Interchange Format**

**Partition VI: Annexes**

# Partition I: Concepts and Architecture

## Table of Contents

<b>Foreword</b>		<b>vii</b>
<b>1</b>	<b>Scope</b>	<b>1</b>
<b>2</b>	<b>Conformance</b>	<b>2</b>
<b>3</b>	<b>Normative references</b>	<b>3</b>
<b>4</b>	<b>Conventions</b>	<b>4</b>
4.1	Organization	4
4.2	Informative text	4
<b>5</b>	<b>Terms and definitions</b>	<b>5</b>
<b>6</b>	<b>Overview of the Common Language Infrastructure</b>	<b>9</b>
6.1	Relationship to type safety	9
6.2	Relationship to managed metadata-driven execution	10
6.2.1	Managed code	10
6.2.2	Managed data	11
6.2.3	Summary	11
<b>7</b>	<b>Common Language Specification</b>	<b>12</b>
7.1	Introduction	12
7.2	Views of CLS compliance	12
7.2.1	CLS framework	12
7.2.2	CLS consumer	13
7.2.3	CLS extender	13
7.3	CLS compliance	14
7.3.1	Marking items as CLS-compliant	15
<b>8</b>	<b>Common Type System</b>	<b>16</b>
8.1	Relationship to object-oriented programming	18
8.2	Values and types	18
8.2.1	Value types and reference types	18
8.2.2	Built-in value and reference types	19
8.2.3	Classes, interfaces, and objects	19
8.2.4	Boxing and unboxing of values	20

8.2.5	Identity and equality of values	21
8.3	Locations	22
8.3.1	Assignment-compatible locations	22
8.3.2	Coercion	22
8.3.3	Casting	22
8.4	Type members	22
8.4.1	Fields, array elements, and values	23
8.4.2	Methods	23
8.4.3	Static fields and static methods	23
8.4.4	Virtual methods	23
8.5	Naming	24
8.5.1	Valid names	24
8.5.2	Assemblies and scoping	24
8.5.3	Visibility, accessibility, and security	26
8.6	Contracts	28
8.6.1	Signatures	29
8.7	Assignment compatibility	32
8.8	Type safety and verification	33
8.9	Type definers	33
8.9.1	Array types	34
8.9.2	Unmanaged pointer types	35
8.9.3	Delegates	35
8.9.4	Interface type definition	36
8.9.5	Class type definition	37
8.9.6	Object type definitions	38
8.9.7	Value type definition	40
8.9.8	Type inheritance	41
8.9.9	Object type inheritance	41
8.9.10	Value type inheritance	41
8.9.11	Interface type derivation	42
8.10	Member inheritance	42
8.10.1	Field inheritance	42
8.10.2	Method inheritance	42
8.10.3	Property and event inheritance	42
8.10.4	Hiding, overriding, and layout	43
8.11	Member definitions	44
8.11.1	Method definitions	44
8.11.2	Field definitions	44

8.11.3	Property definitions	45
8.11.4	Event definitions	46
8.11.5	Nested type definitions	46
<b>9</b>	<b>Metadata</b>	<b>47</b>
9.1	Components and assemblies	47
9.2	Accessing metadata	47
9.2.1	Metadata tokens	47
9.2.2	Member signatures in metadata	48
9.3	Unmanaged code	48
9.4	Method implementation metadata	48
9.5	Class layout	48
9.6	Assemblies: name scopes for types	49
9.7	Metadata extensibility	50
9.8	Globals, imports, and exports	51
9.9	Scoped statics	51
<b>10</b>	<b>Name and type rules for the Common Language Specification</b>	<b>52</b>
10.1	Identifiers	52
10.2	Overloading	52
10.3	Operator overloading	53
10.3.1	Unary operators	53
10.3.2	Binary operators	54
10.3.3	Conversion operators	55
10.4	Naming patterns	56
10.5	Exceptions	56
10.6	Custom attributes	57
10.7	Generic types and methods	57
10.7.1	Nested type parameter re-declaration	57
10.7.2	Type names and arity encoding	58
10.7.3	Type constraint re-declaration	60
10.7.4	Constraint type restrictions	60
10.7.5	Frameworks and accessibility of nested types	60
10.7.6	Frameworks and abstract or virtual methods	61
<b>11</b>	<b>Collected Common Language Specification rules</b>	<b>62</b>
<b>12</b>	<b>Virtual Execution System</b>	<b>65</b>
12.1	Supported data types	65

12.1.1	Native size: native int, native unsigned int, O and &	66
12.1.2	Handling of short integer data types	67
12.1.3	Handling of floating-point data types	67
12.1.4	CIL instructions and numeric types	69
12.1.5	CIL instructions and pointer types	70
12.1.6	Aggregate data	71
12.2	Module information	74
12.3	Machine state	74
12.3.1	The global state	74
12.3.2	Method state	75
12.4	Control flow	78
12.4.1	Method calls	79
12.4.2	Exception handling	82
12.5	Proxies and remoting	92
12.6	Memory model and optimizations	93
12.6.1	The memory store	93
12.6.2	Alignment	93
12.6.3	Byte ordering	93
12.6.4	Optimization	93
12.6.5	Locks and threads	94
12.6.6	Atomic reads and writes	95
12.6.7	Volatile reads and writes	95
12.6.8	Other memory model issues	96
<b>13</b>	<b>Index</b>	<b>97</b>

# Partition II: Metadata Definition and Semantics

## Table of contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Overview</b>	<b>2</b>
<b>3</b>	<b>Validation and verification</b>	<b>3</b>
<b>4</b>	<b>Introductory examples</b>	<b>4</b>
4.1	“Hello world!”	4
4.2	Other examples	4
<b>5</b>	<b>General syntax</b>	<b>5</b>
5.1	General syntax notation	5
5.2	Basic syntax categories	5
5.3	Identifiers	6
5.4	Labels and lists of labels	7
5.5	Lists of hex bytes	7
5.6	Floating-point numbers	7
5.7	Source line information	8
5.8	File names	8
5.9	Attributes and metadata	8
5.10	<i>ilasm</i> source files	9
<b>6</b>	<b>Assemblies, manifests and modules</b>	<b>10</b>
6.1	Overview of modules, assemblies, and files	10
6.2	Defining an assembly	11
6.2.1	Information about the assembly ( <i>AsmDecl</i> )	12
6.2.2	Manifest resources	14
6.2.3	Associating files with an assembly	14
6.3	Referencing assemblies	14
6.4	Declaring modules	15
6.5	Referencing modules	16
6.6	Declarations inside a module or assembly	16
6.7	Exported type definitions	16
<b>7</b>	<b>Types and signatures</b>	<b>18</b>
7.1	Types	18

7.1.1	modreq and modopt	19
7.1.2	pinned	20
7.2	Built-in types	20
7.3	References to user-defined types ( <i>TypeReference</i> )	20
7.4	Native data types	21
<b>8</b>	<b>Visibility, accessibility and hiding</b>	<b>23</b>
8.1	Visibility of top-level types and accessibility of nested types	23
8.2	Accessibility	23
8.3	Hiding	23
<b>9</b>	<b>Generics</b>	<b>24</b>
9.1	Generic type definitions	25
9.2	Generics and recursive inheritance graphs	25
9.3	Generic method definitions	26
9.4	Instantiating generic types	27
9.5	Generics variance	28
9.6	Assignment compatibility of instantiated types	28
9.7	Validity of member signatures	29
9.8	Signatures and binding	30
9.9	Inheritance and overriding	31
9.10	Explicit method overrides	32
9.11	Constraints on generic parameters	33
9.12	References to members of generic types	34
<b>10</b>	<b>Defining types</b>	<b>35</b>
10.1	Type header ( <i>ClassHeader</i> )	35
10.1.1	Visibility and accessibility attributes	36
10.1.2	Type layout attributes	37
10.1.3	Type semantics attributes	37
10.1.4	Inheritance attributes	38
10.1.5	Interoperation attributes	38
10.1.6	Special handling attributes	38
10.1.7	Generic parameters ( <i>GenPars</i> )	39
10.2	Body of a type definition	43
10.3	Introducing and overriding virtual methods	44
10.3.1	Introducing a virtual method	44
10.3.2	The <code>.override</code> directive	44
10.3.3	Accessibility and overriding	45

10.4	Method implementation requirements	46
10.5	Special members	46
10.5.1	Instance constructor	46
10.5.2	Instance finalizer	47
10.5.3	Type initializer	47
10.6	Nested types	49
10.7	Controlling instance layout	49
10.8	Global fields and methods	50
<b>11</b>	<b>Semantics of classes</b>	<b>52</b>
<b>12</b>	<b>Semantics of interfaces</b>	<b>53</b>
12.1	Implementing interfaces	53
12.2	Implementing virtual methods on interfaces	53
<b>13</b>	<b>Semantics of value types</b>	<b>55</b>
13.1	Referencing value types	56
13.2	Initializing value types	56
13.3	Methods of value types	57
<b>14</b>	<b>Semantics of special types</b>	<b>59</b>
14.1	Vectors	59
14.2	Arrays	59
14.3	Enums	61
14.4	Pointer types	62
14.4.1	Unmanaged pointers	63
14.4.2	Managed pointers	64
14.5	Method pointers	64
14.6	Delegates	65
14.6.1	Delegate signature compatibility	66
14.6.2	Synchronous calls to delegates	67
14.6.3	Asynchronous calls to delegates	68
<b>15</b>	<b>Defining, referencing, and calling methods</b>	<b>70</b>
15.1	Method descriptors	70
15.1.1	Method declarations	70
15.1.2	Method definitions	70
15.1.3	Method references	70
15.1.4	Method implementations	70
15.2	Static, instance, and virtual methods	70

15.3	Calling convention	71
15.4	Defining methods	72
15.4.1	Method body	73
15.4.2	Predefined attributes on methods	76
15.4.3	Implementation attributes of methods	78
15.4.4	Scope blocks	79
15.4.5	vararg methods	79
15.5	Unmanaged methods	80
15.5.1	Method transition thunks	80
15.5.2	Platform invoke	81
15.5.3	Method calls via function pointers	82
15.5.4	Data type marshaling	82
<b>16</b>	<b>Defining and referencing fields</b>	<b>84</b>
16.1	Attributes of fields	84
16.1.1	Accessibility information	85
16.1.2	Field contract attributes	85
16.1.3	Interoperation attributes	85
16.1.4	Other attributes	86
16.2	Field init metadata	86
16.3	Embedding data in a PE file	87
16.3.1	Data declaration	87
16.3.2	Accessing data from the PE file	88
16.4	Initialization of non-literal static data	88
16.4.1	Data known at link time	88
16.5	Data known at load time	89
16.5.1	Data known at run time	89
<b>17</b>	<b>Defining properties</b>	<b>90</b>
<b>18</b>	<b>Defining events</b>	<b>92</b>
<b>19</b>	<b>Exception handling</b>	<b>95</b>
19.1	Protected blocks	95
19.2	Handler blocks	96
19.3	Catch blocks	96
19.4	Filter blocks	96
19.5	Finally blocks	97
19.6	Fault handlers	97

<b>20</b>	<b>Declarative security</b>	<b>98</b>
<b>21</b>	<b>Custom attributes</b>	<b>99</b>
21.1	CLS conventions: custom attribute usage	99
21.2	Attributes used by the CLI	99
21.2.1	Pseudo custom attributes	100
21.2.2	Custom attributes defined by the CLS	101
21.2.3	Custom attributes for security	101
21.2.4	Custom attributes for TLS	102
21.2.5	Custom attributes, various	102
<b>22</b>	<b>Metadata logical format: tables</b>	<b>103</b>
22.1	Metadata validation rules	104
22.2	Assembly : 0x20	105
22.3	AssemblyOS : 0x22	106
22.4	AssemblyProcessor : 0x21	106
22.5	AssemblyRef : 0x23	106
22.6	AssemblyRefOS : 0x25	107
22.7	AssemblyRefProcessor : 0x24	107
22.8	ClassLayout : 0x0F	108
22.9	Constant : 0x0B	110
22.10	CustomAttribute : 0x0C	110
22.11	DeclSecurity : 0x0E	112
22.12	EventMap : 0x12	114
22.13	Event : 0x14	114
22.14	ExportedType : 0x27	116
22.15	Field : 0x04	117
22.16	FieldLayout : 0x10	119
22.17	FieldMarshal : 0x0D	120
22.18	FieldRVA : 0x1D	121
22.19	File : 0x26	121
22.20	GenericParam : 0x2A	122
22.21	GenericParamConstraint : 0x2C	123
22.22	ImplMap : 0x1C	124
22.23	InterfaceImpl : 0x09	125
22.24	ManifestResource : 0x28	125
22.25	MemberRef : 0x0A	126
22.26	MethodDef : 0x06	127
22.27	MethodImpl : 0x19	130

22.28	MethodSemantics : 0x18	131
22.29	MethodSpec : 0x2B	132
22.30	Module : 0x00	133
22.31	ModuleRef : 0x1A	133
22.32	NestedClass : 0x29	134
22.33	Param : 0x08	134
22.34	Property : 0x17	135
22.35	PropertyMap : 0x15	136
22.36	StandAloneSig : 0x11	137
22.37	TypeDef : 0x02	138
22.38	TypeRef : 0x01	141
22.39	TypeSpec : 0x1B	142
<b>23</b>	<b>Metadata logical format: other structures</b>	<b>143</b>
23.1	Bitmasks and flags	143
23.1.1	Values for AssemblyHashAlgorithm	143
23.1.2	Values for AssemblyFlags	143
23.1.3	Values for Culture	143
23.1.4	Flags for events [EventAttributes]	144
23.1.5	Flags for fields [FieldAttributes]	144
23.1.6	Flags for files [FileAttributes]	145
23.1.7	Flags for Generic Parameters [GenericParamAttributes]	145
23.1.8	Flags for ImplMap [PInvokeAttributes]	146
23.1.9	Flags for ManifestResource [ManifestResourceAttributes]	146
23.1.10	Flags for methods [MethodAttributes]	146
23.1.11	Flags for methods [MethodImplAttributes]	147
23.1.12	Flags for MethodSemantics [MethodSemanticsAttributes]	148
23.1.13	Flags for params [ParamAttributes]	148
23.1.14	Flags for properties [PropertyAttributes]	148
23.1.15	Flags for types [TypeAttributes]	148
23.1.16	Element types used in signatures	150
23.2	Blobs and signatures	152
23.2.1	MethodDefSig	153
23.2.2	MethodRefSig	154
23.2.3	StandAloneMethodSig	155
23.2.4	FieldSig	157
23.2.5	PropertySig	157
23.2.6	LocalVarSig	157

23.2.7	CustomMod	158
23.2.8	TypeDefOrRefEncoded	158
23.2.9	Constraint	159
23.2.10	Param	159
23.2.11	RetType	159
23.2.12	Type	160
23.2.13	ArrayShape	160
23.2.14	TypeSpec	161
23.2.15	MethodSpec	161
23.2.16	Short form signatures	162
23.3	Custom attributes	162
23.4	Marshalling descriptors	164
<b>24</b>	<b>Metadata physical layout</b>	<b>167</b>
24.1	Fixed fields	167
24.2	File headers	167
24.2.1	Metadata root	167
24.2.2	Stream header	168
24.2.3	#Strings heap	168
24.2.4	#US and #Blob heaps	168
24.2.5	#GUID heap	169
24.2.6	#~ stream	169
<b>25</b>	<b>File format extensions to PE</b>	<b>173</b>
25.1	Structure of the runtime file format	173
25.2	PE headers	173
25.2.1	MS-DOS header	174
25.2.2	PE file header	174
25.2.3	PE optional header	175
25.3	Section headers	177
25.3.1	Import Table and Import Address Table (IAT)	178
25.3.2	Relocations	178
25.3.3	CLI header	179
25.4	Common Intermediate Language physical layout	180
25.4.1	Method header type values	181
25.4.2	Tiny format	181
25.4.3	Fat format	181
25.4.4	Flags for method headers	182
25.4.5	Method data section	182
25.4.6	Exception handling clauses	183
<b>26</b>	<b>Index</b>	<b>184</b>

# Partition III: CIL Instruction Set

## Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Data types	1
1.1.1	Numeric data types	2
1.1.2	Boolean data type	4
1.1.3	Object references	4
1.1.4	Runtime pointer types	4
1.2	Instruction variant table	5
1.2.1	Opcode encodings	6
1.3	Stack transition diagram	11
1.4	English description	11
1.5	Operand type table	11
1.6	Implicit argument coercion	14
1.7	Restrictions on CIL code sequences	16
1.7.1	The instruction stream	16
1.7.2	Valid branch targets	16
1.7.3	Exception ranges	17
1.7.4	Must provide maxstack	17
1.7.5	Backward branch constraints	17
1.7.6	Branch verification constraints	17
1.8	Verifiability and correctness	18
1.8.1	Flow control restrictions for verifiable CIL	18
1.9	Metadata tokens	23
1.10	Exceptions thrown	23
<b>2</b>	<b>Prefixes to instructions</b>	<b>24</b>
2.1	<code>constrained.</code> – (prefix) invoke a member on a value of a variable type	25
2.2	<code>no.</code> – (prefix) possibly skip a fault check	27
2.3	<code>readonly.</code> (prefix) – following instruction returns a controlled-mutability managed pointer	28
2.4	<code>tail.</code> (prefix) – call terminates current method	29
2.5	<code>unaligned.</code> (prefix) – pointer instruction might be unaligned	30
2.6	<code>volatile.</code> (prefix) – pointer reference is volatile	31
<b>3</b>	<b>Base instructions</b>	<b>32</b>
3.1	<code>add</code> – add numeric values	33
3.2	<code>add.ovf.&lt;signed&gt;</code> – add integer values with overflow check	34

3.3	<code>and</code> – bitwise AND	35
3.4	<code>arglist</code> – get argument list	36
3.5	<code>beq.&lt;length&gt;</code> – branch on equal	37
3.6	<code>bge.&lt;length&gt;</code> – branch on greater than or equal to	38
3.7	<code>bge.un.&lt;length&gt;</code> – branch on greater than or equal to, unsigned or unordered	39
3.8	<code>bgt.&lt;length&gt;</code> – branch on greater than	40
3.9	<code>bgt.un.&lt;length&gt;</code> – branch on greater than, unsigned or unordered	41
3.10	<code>ble.&lt;length&gt;</code> – branch on less than or equal to	42
3.11	<code>ble.un.&lt;length&gt;</code> – branch on less than or equal to, unsigned or unordered	43
3.12	<code>blt.&lt;length&gt;</code> – branch on less than	44
3.13	<code>blt.un.&lt;length&gt;</code> – branch on less than, unsigned or unordered	45
3.14	<code>bne.un.&lt;length&gt;</code> – branch on not equal or unordered	46
3.15	<code>br.&lt;length&gt;</code> – unconditional branch	47
3.16	<code>break</code> – breakpoint instruction	48
3.17	<code>brfalse.&lt;length&gt;</code> – branch on false, null, or zero	49
3.18	<code>brtrue.&lt;length&gt;</code> – branch on non-false or non-null	50
3.19	<code>call</code> – call a method	51
3.20	<code>calli</code> – indirect method call	53
3.21	<code>ceq</code> – compare equal	54
3.22	<code>cgt</code> – compare greater than	55
3.23	<code>cgt.un</code> – compare greater than, unsigned or unordered	56
3.24	<code>ckfinite</code> – check for a finite real number	57
3.25	<code>clt</code> – compare less than	58
3.26	<code>clt.un</code> – compare less than, unsigned or unordered	59
3.27	<code>conv.&lt;to type&gt;</code> – data conversion	60
3.28	<code>conv.ovf.&lt;to type&gt;</code> – data conversion with overflow detection	61
3.29	<code>conv.ovf.&lt;to type&gt;.un</code> – unsigned data conversion with overflow detection	62
3.30	<code>cpblk</code> – copy data from memory to memory	63
3.31	<code>div</code> – divide values	64
3.32	<code>div.un</code> – divide integer values, unsigned	65
3.33	<code>dup</code> – duplicate the top value of the stack	66
3.34	<code>endfilter</code> – end exception handling filter clause	67
3.35	<code>endfinally</code> – end the finally or fault clause of an exception block	68
3.36	<code>initblk</code> – initialize a block of memory to a value	69
3.37	<code>jmp</code> – jump to method	70
3.38	<code>ldarg.&lt;length&gt;</code> – load argument onto the stack	71
3.39	<code>ldarga.&lt;length&gt;</code> – load an argument address	72

3.40	ldc.<type> – load numeric constant	73
3.41	ldftn – load method pointer	74
3.42	ldind.<type> – load value indirect onto the stack	75
3.43	ldloc – load local variable onto the stack	77
3.44	ldloca.<length> – load local variable address	78
3.45	ldnull – load a null pointer	79
3.46	leave.<length> – exit a protected region of code	80
3.47	localloc – allocate space in the local dynamic memory pool	81
3.48	mul – multiply values	82
3.49	mul.ovf.<type> – multiply integer values with overflow check	83
3.50	neg – negate	84
3.51	nop – no operation	85
3.52	not – bitwise complement	86
3.53	or – bitwise OR	87
3.54	pop – remove the top element of the stack	88
3.55	rem – compute remainder	89
3.56	rem.un – compute integer remainder, unsigned	90
3.57	ret – return from method	91
3.58	shl – shift integer left	92
3.59	shr – shift integer right	93
3.60	shr.un – shift integer right, unsigned	94
3.61	starg.<length> – store a value in an argument slot	95
3.62	stind.<type> – store value indirect from stack	96
3.63	stloc – pop value from stack to local variable	97
3.64	sub – subtract numeric values	98
3.65	sub.ovf.<type> – subtract integer values, checking for overflow	99
3.66	switch – table switch based on value	100
3.67	xor – bitwise XOR	101
<b>4</b>	<b>Object model instructions</b>	<b>102</b>
4.1	box – convert a boxable value to its boxed form	103
4.2	callvirt – call a method associated, at runtime, with an object	104
4.3	castclass – cast an object to a class	105
4.4	cpobj – copy a value from one address to another	106
4.5	initobj – initialize the value at an address	107
4.6	isinst – test if an object is an instance of a class or interface	108
4.7	ldelem – load element from array	109
4.8	ldelem.<type> – load an element of an array	110

4.9	ldelema – load address of an element of an array	112
4.10	ldfld – load field of an object	113
4.11	ldflda – load field address	114
4.12	ldlen – load the length of an array	115
4.13	ldobj – copy a value from an address to the stack	116
4.14	ldsfld – load static field of a class	117
4.15	ldsflda – load static field address	118
4.16	ldstr – load a literal string	119
4.17	ldtoken – load the runtime representation of a metadata token	120
4.18	ldvirtfn – load a virtual method pointer	121
4.19	mkrefany – push a typed reference on the stack	122
4.20	newarr – create a zero-based, one-dimensional array	123
4.21	newobj – create a new object	124
4.22	refanytype – load the type out of a typed reference	125
4.23	refanyval – load the address out of a typed reference	126
4.24	rethrow – rethrow the current exception	127
4.25	sizeof – load the size, in bytes, of a type	128
4.26	stelem – store element to array	129
4.27	stelem.<type> – store an element of an array	130
4.28	stfld – store into a field of an object	131
4.29	stobj – store a value at an address	132
4.30	stsfld – store a static field of a class	133
4.31	throw – throw an exception	134
4.32	unbox – convert boxed value type to its raw form	135
4.33	unbox.any – convert boxed type to value	136
<b>5</b>	<b>Index</b>	<b>137</b>

# Partition IV:

## Profiles and Libraries

### Table of contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Libraries and Profiles</b>	<b>2</b>
2.1	Libraries	2
2.2	Profiles	2
2.3	The relationship between Libraries and Profiles	3
<b>3</b>	<b>The Standard Profiles</b>	<b>4</b>
3.1	The Kernel Profile	4
3.2	The Compact Profile	4
<b>4</b>	<b>Kernel Profile feature requirements</b>	<b>5</b>
4.1	Features excluded from the Kernel Profile	5
4.1.1	Floating point	5
4.1.2	Non-vector arrays	5
4.1.3	Reflection	5
4.1.4	Application domains	6
4.1.5	Remoting	6
4.1.6	Vararg	6
4.1.7	Frame growth	6
4.1.8	Filtered exceptions	6
<b>5</b>	<b>The standard libraries</b>	<b>7</b>
5.1	General comments	7
5.2	Runtime infrastructure library	7
5.3	Base Class Library (BCL)	7
5.4	Network library	7
5.5	Reflection library	7
5.6	XML library	7
5.7	Extended numerics library	8
5.8	Extended array library	8
5.9	Vararg library	8
5.10	Parallel library	8
<b>6</b>	<b>Implementation-specific modifications to the system libraries</b>	<b>10</b>
<b>7</b>	<b>The XML specification</b>	<b>11</b>

7.1	Semantics	11
7.1.1	Value types as objects	19
7.1.2	Exceptions	19
7.2	XML signature notation issues	19
7.2.1	Serialization	19
7.2.2	Delegates	19
7.2.3	Properties	20
7.2.4	Nested types	20

# Partition V:

## Debug Interchange Format

### Table of contents

<b>1</b>	<b>Portable CILDB files</b>	<b>1</b>
1.1	Encoding of integers	1
1.2	CILDB header	1
1.2.1	Version GUID	1
1.3	Tables and heaps	1
1.3.1	SymConstant table	2
1.3.2	SymDocument table	2
1.3.3	SymMethod table	2
1.3.4	SymSequencePoint table	3
1.3.5	SymScope table	3
1.3.6	SymVariable table	4
1.3.7	SymUsing table	4
1.3.8	SymMisc heap	4
1.3.9	SymString heap	4
1.4	Signatures	4

# Partition VI:

## Annexes

<b>Annex A</b>	<b>Introduction</b>	<b>1</b>
<b>Annex B</b>	<b>Sample programs</b>	<b>2</b>
B.1	Mutually recursive program (with tail calls)	2
B.2	Using value types	3
B.3	Custom attributes	6
B.4	Generics code and metadata	9
B.4.1	ILAsm version	9
B.4.2	C# version	10
B.4.3	Metadata	11
<b>Annex C</b>	<b>CIL assembler implementation</b>	<b>12</b>
C.1	ILAsm keywords	12
C.2	CIL opcode descriptions	15
C.3	Complete grammar	26
C.4	Instruction syntax	41
C.4.1	Top-level instruction syntax	42
C.4.2	Instructions with no operand	42
C.4.3	Instructions that refer to parameters or local variables	44
C.4.4	Instructions that take a single 32-bit integer argument	45
C.4.5	Instructions that take a single 64-bit integer argument	45
C.4.6	Instructions that take a single floating-point argument	45
C.4.7	Branch instructions	45
C.4.8	Instructions that take a method as an argument	46
C.4.9	Instructions that take a field of a class as an argument	46
C.4.10	Instructions that take a type as an argument	47
C.4.11	Instructions that take a string as an argument	47
C.4.12	Instructions that take a signature as an argument	47
C.4.13	Instructions that take a metadata token as an argument	47
C.4.14	Switch instruction	48
<b>Annex D</b>	<b>Class library design guidelines</b>	<b>49</b>
<b>Annex E</b>	<b>Portability considerations</b>	<b>50</b>
E.1	Uncontrollable behavior	50
E.2	Language- and compiler-controllable behavior	50
E.3	Programmer-controllable behavior	50
<b>Annex F</b>	<b>Imprecise faults</b>	<b>52</b>
F.1	Instruction reordering	52

ISO/IEC 23271:2006(E)		
F.2	Inlining	52
F.3	Finally handlers still guaranteed once a try block is entered	52
F.4	Interleaved calls	53
F.4.1	Rejected notions for fencing	53
F.5	Examples	54
F.5.1	Hoisting checks out of a loop	54
F.5.2	Vectorizing a loop	54
F.5.3	Autothreading a loop	55
<b>Annex G</b>	<b>Parallel library</b>	<b>56</b>
G.1	Considerations	56
G.2	ParallelFor	56
G.3	ParallelForEach	57
G.4	ParallelWhile	57
G.5	Debugging	57