

Standard ECMA-335

6th Edition / June 2012

Common Language Infrastructure (CLI) Partitions I to VI

Standard



COPYRIGHT PROTECTED DOCUMENT

This Ecma Standard has been adopted by the General Assembly of June 2012.

"COPYRIGHT NOTICE

© <2012> Ecma International

This document may be copied, published and distributed to others, and certain derivative works of it may be prepared, copied, published, and distributed, in whole or in part, provided that the above copyright notice and this Copyright License and Disclaimer are included on all such copies and derivative works. The only derivative works that are permissible under this Copyright License and Disclaimer are:

- (i) works which incorporate all or portion of this document for the purpose of providing commentary or explanation (such as an annotated version of the document),*
- (ii) works which incorporate all or portion of this document for the purpose of incorporating features that provide accessibility,*
- (iii) translations of this document into languages other than English and into different formats and*
- (iv) works by making use of this specification in standard conformant products by implementing (e.g. by copy and paste wholly or partly) the functionality therein.*

However, the content of this document itself may not be modified in any way, including by removing the copyright notice or references to Ecma International, except as required to translate it into languages other than English or into a different format.

The official version of an Ecma International document is the English language version on the Ecma International website. In the event of discrepancies between a translated version and the official version, the official version shall govern.

The limited permissions granted above are perpetual and will not be revoked by Ecma International or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and ECMA INTERNATIONAL DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE "

**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: Binary Formats
Partition VI: Annexes

Table of Contents

Foreword	xxi
I.1 Scope	1
I.2 Conformance	2
I.3 Normative references	3
I.4 Conventions	5
I.4.1 Organization	5
I.4.2 Informative text	5
I.5 Terms and definitions	6
I.6 Overview of the Common Language Infrastructure	9
I.6.1 Relationship to type safety	9
I.6.2 Relationship to managed metadata-driven execution	10
I.6.2.1 Managed code	10
I.6.2.2 Managed data	11
I.6.2.3 Summary	11
I.7 Common Language Specification	12
I.7.1 Introduction	12
I.7.2 Views of CLS compliance	12
I.7.2.1 CLS framework	12
I.7.2.2 CLS consumer	13
I.7.2.3 CLS extender	13
I.7.3 CLS compliance	14
I.7.3.1 Marking items as CLS-compliant	14
I.8 Common Type System	16
I.8.1 Relationship to object-oriented programming	19
I.8.2 Values and types	19
I.8.2.1 Value types and reference types	19
I.8.2.2 Built-in value and reference types	20
I.8.2.3 Classes, interfaces, and objects	21
I.8.2.4 Boxing and unboxing of values	21
I.8.2.5 Identity and equality of values	22
I.8.3 Locations	23
I.8.3.1 Assignment-compatible locations	23
I.8.3.2 Coercion	23

I.8.3.3	Casting	24
I.8.4	Type members	24
I.8.4.1	Fields, array elements, and values	24
I.8.4.2	Methods	24
I.8.4.3	Static fields and static methods	25
I.8.4.4	Virtual methods	25
I.8.5	Naming	25
I.8.5.1	Valid names	25
I.8.5.2	Assemblies and scoping	26
I.8.5.3	Visibility, accessibility, and security	27
I.8.6	Contracts	29
I.8.6.1	Signatures	30
I.8.7	Assignment compatibility	34
I.8.7.1	Assignment compatibility for signature types	37
I.8.7.2	Assignment compatibility for location types	38
I.8.7.3	General assignment compatibility	38
I.8.8	Type safety and verification	39
I.8.9	Type definers	39
I.8.9.1	Array types	40
I.8.9.2	Unmanaged pointer types	41
I.8.9.3	Delegates	41
I.8.9.4	Interface type definition	42
I.8.9.5	Class type definition	43
I.8.9.6	Object type definitions	44
I.8.9.7	Value type definition	47
I.8.9.8	Type inheritance	47
I.8.9.9	Object type inheritance	47
I.8.9.10	Value type inheritance	48
I.8.9.11	Interface type derivation	48
I.8.10	Member inheritance	48
I.8.10.1	Field inheritance	48
I.8.10.2	Method inheritance	48
I.8.10.3	Property and event inheritance	49
I.8.10.4	Hiding, overriding, and layout	49
I.8.11	Member definitions	50
I.8.11.1	Method definitions	50
I.8.11.2	Field definitions	51
I.8.11.3	Property definitions	51
I.8.11.4	Event definitions	52
I.8.11.5	Nested type definitions	52

I.9	Metadata	53
I.9.1	Components and assemblies	53
I.9.2	Accessing metadata	53
I.9.2.1	Metadata tokens	54
I.9.2.2	Member signatures in metadata	54
I.9.3	Unmanaged code	54
I.9.4	Method implementation metadata	54
I.9.5	Class layout	55
I.9.6	Assemblies: name scopes for types	55
I.9.7	Metadata extensibility	56
I.9.8	Globals, imports, and exports	57
I.9.9	Scoped statics	58
I.10	Name and type rules for the Common Language Specification	59
I.10.1	Identifiers	59
I.10.2	Overloading	59
I.10.3	Operator overloading	60
I.10.3.1	Unary operators	60
I.10.3.2	Binary operators	61
I.10.3.3	Conversion operators	62
I.10.4	Naming patterns	62
I.10.5	Exceptions	63
I.10.6	Custom attributes	63
I.10.7	Generic types and methods	64
I.10.7.1	Nested type parameter re-declaration	64
I.10.7.2	Type names and arity encoding	65
I.10.7.3	Type constraint re-declaration	66
I.10.7.4	Constraint type restrictions	67
I.10.7.5	Frameworks and accessibility of nested types	67
I.10.7.6	Frameworks and abstract or virtual methods	68
I.11	Collected Common Language Specification rules	69
I.12	Virtual Execution System	72
I.12.1	Supported data types	72
I.12.1.1	Native size: native int, native unsigned int, O and &	73
I.12.1.2	Handling of short integer data types	74
I.12.1.3	Handling of floating-point data types	75
I.12.1.4	CIL instructions and numeric types	76
I.12.1.5	CIL instructions and pointer types	77
I.12.1.6	Aggregate data	78

I.12.2	Module information	81
I.12.3	Machine state	81
I.12.3.1	The global state	81
I.12.3.2	Method state	82
I.12.4	Control flow	85
I.12.4.1	Method calls	86
I.12.4.2	Exception handling	89
I.12.5	Proxies and remoting	99
I.12.6	Memory model and optimizations	100
I.12.6.1	The memory store	100
I.12.6.2	Alignment	100
I.12.6.3	Byte ordering	100
I.12.6.4	Optimization	100
I.12.6.5	Locks and threads	101
I.12.6.6	Atomic reads and writes	102
I.12.6.7	Volatile reads and writes	102
I.12.6.8	Other memory model issues	103
II.1	Introduction	105
II.2	Overview	106
II.3	Validation and verification	107
II.4	Introductory examples	108
II.4.1	“Hello world!”	108
II.4.2	Other examples	108
II.5	General syntax	109
II.5.1	General syntax notation	109
II.5.2	Basic syntax categories	109
II.5.3	Identifiers	110
II.5.4	Labels and lists of labels	111
II.5.5	Lists of hex bytes	111
II.5.6	Floating-point numbers	111
II.5.7	Source line information	112
II.5.8	File names	112
II.5.9	Attributes and metadata	112
II.5.10	<i>ilasm</i> source files	112
II.6	Assemblies, manifests and modules	114
II.6.1	Overview of modules, assemblies, and files	114
II.6.2	Defining an assembly	115
II.6.2.1	Information about the assembly (<i>AsmDecl</i>)	115

II.6.2.2	Manifest resources	118
II.6.2.3	Associating files with an assembly	118
II.6.3	Referencing assemblies	118
II.6.4	Declaring modules	119
II.6.5	Referencing modules	120
II.6.6	Declarations inside a module or assembly	120
II.6.7	Exported type definitions	120
II.6.8	Type forwarders	121
II.7	Types and signatures	122
II.7.1	Types	122
II.7.1.1	modreq and modopt	123
II.7.1.2	pinned	123
II.7.2	Built-in types	124
II.7.3	References to user-defined types (<i>TypeReference</i>)	124
II.7.4	Native data types	125
II.8	Visibility, accessibility and hiding	127
II.8.1	Visibility of top-level types and accessibility of nested types	127
II.8.2	Accessibility	127
II.8.3	Hiding	127
II.9	Generics	128
II.9.1	Generic type definitions	128
II.9.2	Generics and recursive inheritance graphs	129
II.9.3	Generic method definitions	130
II.9.4	Instantiating generic types	131
II.9.5	Generics variance	132
II.9.6	Assignment compatibility of instantiated types	132
II.9.7	Validity of member signatures	133
II.9.8	Signatures and binding	134
II.9.9	Inheritance and overriding	135
II.9.10	Explicit method overrides	136
II.9.11	Constraints on generic parameters	137
II.9.12	References to members of generic types	138
II.10	Defining types	139
II.10.1	Type header (<i>ClassHeader</i>)	139
II.10.1.1	Visibility and accessibility attributes	140
II.10.1.2	Type layout attributes	141
II.10.1.3	Type semantics attributes	141
II.10.1.4	Inheritance attributes	142

II.10.1.5	Interoperation attributes	142
II.10.1.6	Special handling attributes	142
II.10.1.7	Generic parameters (<i>GenPars</i>)	143
II.10.2	Body of a type definition	146
II.10.3	Introducing and overriding virtual methods	147
II.10.3.1	Introducing a virtual method	147
II.10.3.2	The <code>.override</code> directive	147
II.10.3.3	Accessibility and overriding	148
II.10.3.4	Impact of overrides on derived classes	149
II.10.4	Method implementation requirements	150
II.10.5	Special members	150
II.10.5.1	Instance constructor	150
II.10.5.2	Instance finalizer	151
II.10.5.3	Type initializer	151
II.10.6	Nested types	153
II.10.7	Controlling instance layout	153
II.10.8	Global fields and methods	154
II.11	Semantics of classes	156
II.12	Semantics of interfaces	157
II.12.1	Implementing interfaces	157
II.12.2	Implementing virtual methods on interfaces	157
II.12.2.1	Interface Implementation Examples	159
II.13	Semantics of value types	162
II.13.1	Referencing value types	163
II.13.2	Initializing value types	163
II.13.3	Methods of value types	164
II.14	Semantics of special types	166
II.14.1	Vectors	166
II.14.2	Arrays	166
II.14.3	Enums	168
II.14.4	Pointer types	169
II.14.4.1	Unmanaged pointers	170
II.14.4.2	Managed pointers	171
II.14.5	Method pointers	171
II.14.6	Delegates	172
II.14.6.1	Delegate signature compatibility	173
II.14.6.2	Synchronous calls to delegates	174
II.14.6.3	Asynchronous calls to delegates	175

II.15	Defining, referencing, and calling methods	177
II.15.1	Method descriptors	177
II.15.1.1	Method declarations	177
II.15.1.2	Method definitions	177
II.15.1.3	Method references	177
II.15.1.4	Method implementations	177
II.15.2	Static, instance, and virtual methods	177
II.15.3	Calling convention	178
II.15.4	Defining methods	179
II.15.4.1	Method body	180
II.15.4.2	Predefined attributes on methods	182
II.15.4.3	Implementation attributes of methods	184
II.15.4.4	Scope blocks	186
II.15.4.5	vararg methods	186
II.15.5	Unmanaged methods	187
II.15.5.1	Method transition thunks	187
II.15.5.2	Platform invoke	188
II.15.5.3	Method calls via function pointers	189
II.15.5.4	Data type marshaling	189
II.16	Defining and referencing fields	190
II.16.1	Attributes of fields	190
II.16.1.1	Accessibility information	191
II.16.1.2	Field contract attributes	191
II.16.1.3	Interoperation attributes	191
II.16.1.4	Other attributes	192
II.16.2	Field init metadata	192
II.16.3	Embedding data in a PE file	193
II.16.3.1	Data declaration	193
II.16.3.2	Accessing data from the PE file	194
II.16.4	Initialization of non-literal static data	194
II.16.4.1	Data known at link time	194
II.16.5	Data known at load time	195
II.16.5.1	Data known at run time	195
II.17	Defining properties	196
II.18	Defining events	198
II.19	Exception handling	201
II.19.1	Protected blocks	201
II.19.2	Handler blocks	201

II.19.3	Catch blocks	202
II.19.4	Filter blocks	202
II.19.5	Finally blocks	203
II.19.6	Fault handlers	203
II.20	Declarative security	204
II.21	Custom attributes	205
II.21.1	CLS conventions: custom attribute usage	205
II.21.2	Attributes used by the CLI	205
II.21.2.1	Pseudo custom attributes	206
II.21.2.2	Custom attributes defined by the CLS	207
II.21.2.3	Custom attributes for security	207
II.21.2.4	Custom attributes for TLS	207
II.21.2.5	Custom attributes, various	208
II.22	Metadata logical format: tables	209
II.22.1	Metadata validation rules	210
II.22.2	Assembly : 0x20	211
II.22.3	AssemblyOS : 0x22	212
II.22.4	AssemblyProcessor : 0x21	212
II.22.5	AssemblyRef : 0x23	212
II.22.6	AssemblyRefOS : 0x25	213
II.22.7	AssemblyRefProcessor : 0x24	213
II.22.8	ClassLayout : 0x0F	214
II.22.9	Constant : 0x0B	216
II.22.10	CustomAttribute : 0x0C	216
II.22.11	DeclSecurity : 0x0E	218
II.22.12	EventMap : 0x12	220
II.22.13	Event : 0x14	220
II.22.14	ExportedType : 0x27	222
II.22.15	Field : 0x04	223
II.22.16	FieldLayout : 0x10	225
II.22.17	FieldMarshal : 0x0D	226
II.22.18	FieldRVA : 0x1D	227
II.22.19	File : 0x26	227
II.22.20	GenericParam : 0x2A	228
II.22.21	GenericParamConstraint : 0x2C	229
II.22.22	ImplMap : 0x1C	230
II.22.23	InterfaceImpl : 0x09	231
II.22.24	ManifestResource : 0x28	231
II.22.25	MemberRef : 0x0A	232

II.22.26	MethodDef : 0x06	233
II.22.27	MethodImpl : 0x19	236
II.22.28	MethodSemantics : 0x18	237
II.22.29	MethodSpec : 0x2B	238
II.22.30	Module : 0x00	239
II.22.31	ModuleRef : 0x1A	239
II.22.32	NestedClass : 0x29	240
II.22.33	Param : 0x08	240
II.22.34	Property : 0x17	241
II.22.35	PropertyMap : 0x15	242
II.22.36	StandAloneSig : 0x11	243
II.22.37	TypeDef : 0x02	243
II.22.38	TypeRef : 0x01	247
II.22.39	TypeSpec : 0x1B	248
II.23	Metadata logical format: other structures	249
II.23.1	Bitmasks and flags	249
II.23.1.1	Values for AssemblyHashAlgorithm	249
II.23.1.2	Values for AssemblyFlags	249
II.23.1.3	Values for Culture	249
II.23.1.4	Flags for events [EventAttributes]	250
II.23.1.5	Flags for fields [FieldAttributes]	250
II.23.1.6	Flags for files [FileAttributes]	251
II.23.1.7	Flags for Generic Parameters [GenericParamAttributes]	251
II.23.1.8	Flags for ImplMap [PInvokeAttributes]	251
II.23.1.9	Flags for ManifestResource [ManifestResourceAttributes]	252
II.23.1.10	Flags for methods [MethodAttributes]	252
II.23.1.11	Flags for methods [MethodImplAttributes]	253
II.23.1.12	Flags for MethodSemantics [MethodSemanticsAttributes]	253
II.23.1.13	Flags for params [ParamAttributes]	253
II.23.1.14	Flags for properties [PropertyAttributes]	254
II.23.1.15	Flags for types [TypeAttributes]	254
II.23.1.16	Element types used in signatures	255
II.23.2	Blobs and signatures	257
II.23.2.1	MethodDefSig	259
II.23.2.2	MethodRefSig	260
II.23.2.3	StandAloneMethodSig	261
II.23.2.4	FieldSig	262
II.23.2.5	PropertySig	262
II.23.2.6	LocalVarSig	263

II.23.2.7	CustomMod	263
II.23.2.8	TypeDefOrRefOrSpecEncoded	264
II.23.2.9	Constraint	264
II.23.2.10	Param	264
II.23.2.11	RetType	265
II.23.2.12	Type	265
II.23.2.13	ArrayShape	265
II.23.2.14	TypeSpec	266
II.23.2.15	MethodSpec	266
II.23.2.16	Short form signatures	267
II.23.3	Custom attributes	267
II.23.4	Marshalling descriptors	269
II.24	Metadata physical layout	271
II.24.1	Fixed fields	271
II.24.2	File headers	271
II.24.2.1	Metadata root	271
II.24.2.2	Stream header	272
II.24.2.3	#Strings heap	272
II.24.2.4	#US and #Blob heaps	272
II.24.2.5	#GUID heap	272
II.24.2.6	#~ stream	273
II.25	File format extensions to PE	277
II.25.1	Structure of the runtime file format	277
II.25.2	PE headers	277
II.25.2.1	MS-DOS header	278
II.25.2.2	PE file header	278
II.25.2.3	PE optional header	279
II.25.3	Section headers	281
II.25.3.1	Import Table and Import Address Table (IAT)	282
II.25.3.2	Relocations	282
II.25.3.3	CLI header	283
II.25.4	Common Intermediate Language physical layout	284
II.25.4.1	Method header type values	285
II.25.4.2	Tiny format	285
II.25.4.3	Fat format	285
II.25.4.4	Flags for method headers	285
II.25.4.5	Method data section	286
II.25.4.6	Exception handling clauses	286
III.1	Introduction	290

III.1.1	Data types	290
III.1.1.1	Numeric data types	291
III.1.1.2	Boolean data type	293
III.1.1.3	Character data type	293
III.1.1.4	Object references	293
III.1.1.5	Runtime pointer types	293
III.1.2	Instruction variant table	295
III.1.2.1	Opcode encodings	295
III.1.3	Stack transition diagram	301
III.1.4	English description	302
III.1.5	Operand type table	302
III.1.6	Implicit argument coercion	305
III.1.7	Restrictions on CIL code sequences	306
III.1.7.1	The instruction stream	307
III.1.7.2	Valid branch targets	307
III.1.7.3	Exception ranges	307
III.1.7.4	Must provide maxstack	308
III.1.7.5	Backward branch constraints	308
III.1.7.6	Branch verification constraints	308
III.1.8	Verifiability and correctness	308
III.1.8.1	Flow control restrictions for verifiable CIL	309
III.1.9	Metadata tokens	313
III.1.10	Exceptions thrown	314
III.2	Prefixes to instructions	315
III.2.1	constrained. – (prefix) invoke a member on a value of a variable type	316
III.2.2	no. – (prefix) possibly skip a fault check	318
III.2.3	readonly. (prefix) – following instruction returns a controlled-mutability managed pointer	319
III.2.4	tail. (prefix) – call terminates current method	320
III.2.5	unaligned. (prefix) – pointer instruction might be unaligned	321
III.2.6	volatile. (prefix) – pointer reference is volatile	322
III.3	Base instructions	323
III.3.1	add – add numeric values	324
III.3.2	add.ovf.<signed> – add integer values with overflow check	325
III.3.3	and – bitwise AND	326
III.3.4	arglist – get argument list	327
III.3.5	beq.<length> – branch on equal	328
III.3.6	bge.<length> – branch on greater than or equal to	329

III.3.7	bge.un.<length> – branch on greater than or equal to, unsigned or unordered	330
III.3.8	bgt.<length> – branch on greater than	331
III.3.9	bgt.un.<length> – branch on greater than, unsigned or unordered	332
III.3.10	ble.<length> – branch on less than or equal to	333
III.3.11	ble.un.<length> – branch on less than or equal to, unsigned or unordered	334
III.3.12	blt.<length> – branch on less than	335
III.3.13	blt.un.<length> – branch on less than, unsigned or unordered	336
III.3.14	bne.un.<length> – branch on not equal or unordered	337
III.3.15	br.<length> – unconditional branch	338
III.3.16	break – breakpoint instruction	339
III.3.17	brfalse.<length> – branch on false, null, or zero	340
III.3.18	brtrue.<length> – branch on non-false or non-null	341
III.3.19	call – call a method	342
III.3.20	calli – indirect method call	344
III.3.21	ceq – compare equal	346
III.3.22	cgt – compare greater than	347
III.3.23	cgt.un – compare greater than, unsigned or unordered	348
III.3.24	ckfinite – check for a finite real number	349
III.3.25	clt – compare less than	350
III.3.26	clt.un – compare less than, unsigned or unordered	351
III.3.27	conv.<to type> – data conversion	352
III.3.28	conv.ovf.<to type> – data conversion with overflow detection	353
III.3.29	conv.ovf.<to type>.un – unsigned data conversion with overflow detection	354
III.3.30	cpblk – copy data from memory to memory	355
III.3.31	div – divide values	356
III.3.32	div.un – divide integer values, unsigned	357
III.3.33	dup – duplicate the top value of the stack	358
III.3.34	endfilter – end exception handling filter clause	359
III.3.35	endfinally – end the finally or fault clause of an exception block	360
III.3.36	initblk – initialize a block of memory to a value	361
III.3.37	jmp – jump to method	362
III.3.38	ldarg.<length> – load argument onto the stack	363
III.3.39	ldarga.<length> – load an argument address	364
III.3.40	ldc.<type> – load numeric constant	365
III.3.41	ldftn – load method pointer	366
III.3.42	ldind.<type> – load value indirect onto the stack	367

III.3.43	ldloc – load local variable onto the stack	369
III.3.44	ldloca.<length> – load local variable address	370
III.3.45	ldnull – load a null pointer	371
III.3.46	leave.<length> – exit a protected region of code	372
III.3.47	localloc – allocate space in the local dynamic memory pool	373
III.3.48	mul – multiply values	374
III.3.49	mul.ovf.<type> – multiply integer values with overflow check	375
III.3.50	neg – negate	376
III.3.51	nop – no operation	377
III.3.52	not – bitwise complement	378
III.3.53	or – bitwise OR	379
III.3.54	pop – remove the top element of the stack	380
III.3.55	rem – compute remainder	381
III.3.56	rem.un – compute integer remainder, unsigned	382
III.3.57	ret – return from method	383
III.3.58	shl – shift integer left	384
III.3.59	shr – shift integer right	385
III.3.60	shr.un – shift integer right, unsigned	386
III.3.61	starg.<length> – store a value in an argument slot	387
III.3.62	stind.<type> – store value indirect from stack	388
III.3.63	stloc – pop value from stack to local variable	389
III.3.64	sub – subtract numeric values	390
III.3.65	sub.ovf.<type> – subtract integer values, checking for overflow 391	
III.3.66	switch – table switch based on value	392
III.3.67	xor – bitwise XOR	393
III.4	Object model instructions	394
III.4.1	box – convert a boxable value to its boxed form	394
III.4.2	callvirt – call a method associated, at runtime, with an object	396
III.4.3	castclass – cast an object to a class	398
III.4.4	cpobj – copy a value from one address to another	399
III.4.5	initobj – initialize the value at an address	400
III.4.6	isinst – test if an object is an instance of a class or interface	401
III.4.7	ldelem – load element from array	402
III.4.8	ldelem.<type> – load an element of an array	403
III.4.9	ldelema – load address of an element of an array	405
III.4.10	ldfld – load field of an object	406
III.4.11	ldflda – load field address	407
III.4.12	ldlen – load the length of an array	408
III.4.13	ldobj – copy a value from an address to the stack	409

III.4.14	ldsfld – load static field of a class	410
III.4.15	ldsfla – load static field address	411
III.4.16	ldstr – load a literal string	412
III.4.17	ldtoken – load the runtime representation of a metadata token	413
III.4.18	ldvirtftn – load a virtual method pointer	414
III.4.19	mkrefany – push a typed reference on the stack	415
III.4.20	newarr – create a zero-based, one-dimensional array	416
III.4.21	newobj – create a new object	417
III.4.22	refanytype – load the type out of a typed reference	420
III.4.23	refanyval – load the address out of a typed reference	421
III.4.24	rethrow – rethrow the current exception	422
III.4.25	sizeof – load the size, in bytes, of a type	423
III.4.26	stelem – store element to array	424
III.4.27	stelem.<type> – store an element of an array	425
III.4.28	stfld – store into a field of an object	427
III.4.29	stobj – store a value at an address	428
III.4.30	stsfld – store a static field of a class	429
III.4.31	throw – throw an exception	430
III.4.32	unbox – convert boxed value type to its raw form	431
III.4.33	unbox.any – convert boxed type to value	432
IV.1	Overview	434
IV.2	Libraries and Profiles	435
IV.2.1	Libraries	435
IV.2.2	Profiles	435
IV.2.3	The relationship between Libraries and Profiles	436
IV.3	The Standard Profiles	437
IV.3.1	The Kernel Profile	437
IV.3.2	The Compact Profile	437
IV.4	Kernel Profile feature requirements	438
IV.4.1	Features excluded from the Kernel Profile	438
IV.4.1.1	Floating point	438
IV.4.1.2	Non-vector arrays	438
IV.4.1.3	Reflection	438
IV.4.1.4	Application domains	439
IV.4.1.5	Remoting	439
IV.4.1.6	Vararg	439
IV.4.1.7	Frame growth	439
IV.4.1.8	Filtered exceptions	439

IV.5	The standard libraries	440
IV.5.1	General comments	440
IV.5.2	Runtime infrastructure library	440
IV.5.3	Base Class Library (BCL)	440
IV.5.4	Network library	440
IV.5.5	Reflection library	440
IV.5.6	XML library	440
IV.5.7	Extended numerics library	441
IV.5.8	Extended array library	441
IV.5.9	Vararg library	441
IV.5.10	Parallel library	441
IV.6	Implementation-specific modifications to the system libraries	443
IV.7	The XML specification	444
IV.7.1	Semantics	444
IV.7.1.1	Value types as objects	452
IV.7.1.2	Exceptions	452
IV.7.2	XML signature notation issues	452
IV.7.2.1	Serialization	452
IV.7.2.2	Delegates	452
IV.7.2.3	Properties	453
IV.7.2.4	Nested types	453
V.1	Portable CILDB files	455
V.1.1	Encoding of integers	455
V.1.2	CILDB header	455
V.1.2.1	Version GUID	455
V.1.3	Tables and heaps	455
V.1.3.1	SymConstant table	456
V.1.3.2	SymDocument table	456
V.1.3.3	SymMethod table	456
V.1.3.4	SymSequencePoint table	457
V.1.3.5	SymScope table	457
V.1.3.6	SymVariable table	457
V.1.3.7	SymUsing table	458
V.1.3.8	SymMisc heap	458
V.1.3.9	SymString heap	458
V.1.4	Signatures	458
VI. Annex A	Introduction	461

VI. Annex B	Sample programs	462
VI.B.1	Mutually recursive program (with tail calls)	462
VI.B.2	Using value types	463
VI.B.3	Custom attributes	465
VI.B.4	Generics code and metadata	468
VI.B.4.1	ILAsm version	468
VI.B.4.2	C# version	469
VI.B.4.3	Metadata	469
VI. Annex C	CIL assembler implementation	471
VI.C.1	ILAsm keywords	471
VI.C.2	CIL opcode descriptions	483
VI.C.3	Complete grammar	494
VI.C.4	Instruction syntax	509
VI.C.4.1	Top-level instruction syntax	510
VI.C.4.2	Instructions with no operand	510
VI.C.4.3	Instructions that refer to parameters or local variables	511
VI.C.4.4	Instructions that take a single 32-bit integer argument	512
VI.C.4.5	Instructions that take a single 64-bit integer argument	512
VI.C.4.6	Instructions that take a single floating-point argument	512
VI.C.4.7	Branch instructions	513
VI.C.4.8	Instructions that take a method as an argument	513
VI.C.4.9	Instructions that take a field of a class as an argument	513
VI.C.4.10	Instructions that take a type as an argument	513
VI.C.4.11	Instructions that take a string as an argument	514
VI.C.4.12	Instructions that take a signature as an argument	514
VI.C.4.13	Instructions that take a metadata token as an argument	514
VI.C.4.14	Switch instruction	515
VI. Annex D	Class library design guidelines	516
VI. Annex E	Portability considerations	517
VI.E.1	Uncontrollable behavior	517
VI.E.2	Language- and compiler-controllable behavior	517
VI.E.3	Programmer-controllable behavior	517
VI. Annex F	Imprecise faults	519
VI.F.1	Instruction reordering	519
VI.F.2	Inlining	519
VI.F.3	Finally handlers still guaranteed once a try block is entered	519
VI.F.4	Interleaved calls	520
VI.F.4.1	Rejected notions for fencing	520

VI.F.5	Examples	520
VI.F.5.1	Hoisting checks out of a loop	521
VI.F.5.2	Vectorizing a loop	521
VI.F.5.3	Autothreading a loop	521
VI. Annex G	Parallel library	523
VI.G.1	Considerations	523
VI.G.2	ParallelFor	523
VI.G.3	ParallelForEach	523
VI.G.4	ParallelWhile	524
VI.G.5	Debugging	524
Index		525