



Standardizing Information and Communication Systems

---

---

---

**Common Language Infrastructure (CLI)**  
**Part 4: Base Class Library**

---

---

**Draft 01 – October 2000**

This contribution is being provided “AS IS”, and the SPONSORS EXPRESSLY DISCLAIM ANY AND ALL WARRANTIES REGARDING THIS CONTRIBUTION, INCLUDING ANY WARRANTY THAT THIS CONTRIBUTION DOES NOT VIOLATE THE RIGHTS OF OTHERS OR IS FIT FOR A PARTICULAR PURPOSE.

## **Brief History**

This ECMA Standard has been adopted by the ECMA General Assembly of ....



## Table of contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Factoring: Categories, Packages, and Profiles</b>	<b>1</b>
2.1	Factoring And the Base Class Library	1
2.1.1	Missing Base Class	2
2.1.2	Missing Interface	2
2.1.3	Missing Types for Members	2
2.2	Factoring and Profiles	2
2.3	Factoring, File Format, and the Virtual Execution System	3
2.4	Factoring and the Globalization Package	3
2.5	Factoring and Programming Languages	3
<b>3</b>	<b>The Abstract Operating System Category</b>	<b>3</b>
3.1	The Networking Package	3
3.1.1	Impact on File Format, CIL, and Execution Engine	4
3.2	The Threading Package	4
3.2.1	Impact on File Format, CIL, and Execution Engine	4
3.3	The Security Package	4
3.3.1	Impact on File Format, CIL, and Execution Engine	4
3.4	The Standard Input / Output Package	4
3.4.1	Impact on File Format, CIL, and Execution Engine	5
<b>4</b>	<b>The Common Programming Utilities Category</b>	<b>5</b>
4.1	The Collections Package	5
4.1.1	Impact on File Format, CIL, and Execution Engine	5
4.2	The Common Data Types Package	5
4.2.1	Impact on File Format, CIL, and Execution Engine	5
4.3	The Extended Numerics Package	5
4.3.1	Impact on File Format, CIL, and Execution Engine	5
4.4	The Regular Expressions Package	5
4.4.1	Impact on File Format, CIL, and Execution Engine	6
4.5	The Serialization Package	6
4.5.1	Impact on File Format, CIL, and Execution Engine	6
<b>5</b>	<b>The Execution Engine Functionality Category</b>	<b>6</b>
5.1	The Garbage Collection Package	6
5.1.1	Impact on File Format, CIL, and Execution Engine	6
5.2	The Hosting Package	6
5.2.1	Impact on File Format, CIL, and Execution Engine	6
5.3	The Non-CLS Package	6
5.3.1	Impact on File Format, CIL, and Execution Engine	6
5.4	The Reflection Package	7
5.4.1	Impact on File Format, CIL, and Execution Engine	7

5.5	The Unmanaged Code Package	7
5.5.1	Impact on File Format, CIL, and Execution Engine	7
<b>6</b>	<b>The High Level Programming Category</b>	<b>7</b>
6.1	The XML Package	7
6.1.1	Impact on File Format, CIL, and Execution Engine	7
6.2	The Advanced XML Package	7
6.2.1	Impact on File Format, CIL, and Execution Engine	7
6.3	The Asynchronous Programming Package	7
6.3.1	Impact on File Format, CIL, and Execution Engine	8
6.4	The Globalization Package	8
6.4.1	Impact on File Format, CIL, and Execution Engine	8
6.5	The Remote Object Package	8
6.5.1	Impact on File Format, CIL, and Execution Engine	8
<b>7</b>	<b>The Miscellaneous Category</b>	<b>8</b>
7.1	The Kernel Package	8
7.1.1	Impact on File Format, CIL, and Execution Engine	8
7.2	The Basic Language Support Package	9
7.2.1	Impact on File Format, CIL, and Execution Engine	9
7.3	The Development Time Package	9
7.3.1	Impact on File Format, CIL, and Execution Engine	9
<b>8</b>	<b>The Standard Profiles</b>	<b>9</b>
8.1	The Kernel Profile	10
8.2	The Compact Profile	10
8.3	The Complete Profile	10
8.4	Others to be decided	10

## 1 Overview

While compilers are most concerned with issues of file format, instruction set design, and a common type system, the average programmer is most interested in the programming library that is available to them in the language they are using. The Common Language Infrastructure (CLI) specifies a Common Language Specification (CLS) that should be used to define the externally visible aspects (method signatures, etc.) so they can be used from a wide range of programming languages. The CLI Base Class Library (BCL) follows this recommendation so that it can be used from the full range of languages that target the CLI.

The BCL is designed with the following goals in mind:

- Wide reach across programming languages
- Consistent design patterns throughout
- All features required for implementing C# and ECMAScript
- Features on parity with the ANSI C library of 1985
- Features for more recent programming paradigms, notably networking, XML, and remote objects
- Factoring into self-consistent packages with minimal interdependence

This document provides an overview of the BCL and its factoring. A companion document, distributed in XML format, provides details of each class, value type, and interface in the BCL. This XML can be processed using an XSL transform to produce easily browsable information about the BCL. While the normative specification of the BCL is in the XML form, several Web sites that use XSL to process the XML are available on-line to provide a more convenient informative description of the BCL.

Part 5 contains an informative annex describing programming conventions for defining the Base Class Library. These conventions can significantly simplify the use of libraries and implementers are encouraged to follow them when creating additional non-Standard libraries.

## 2 Factoring: Categories, Packages, and Profiles

The BCL is factored on a class basis – that is, a class belongs to one (and only one) of 22 package. These packages are grouped together in two different ways. The first grouping is purely for explanatory purposes, and is the organization used in this specification. In this grouping, each package occurs exactly once in each category.

The second grouping, described in [The Standard Profiles](#), defines sets of packages that provide common levels of functionality and make it easy to specify conformance to this standard. Conforming implementations of the CLI shall implement one of the standard profiles specified in [The Standard Profiles](#). Packages and profiles shall be implemented in their entirety.

### 2.1 Factoring And the Base Class Library

Each package consists of a set of classes, value types, and interfaces. These, in turn, contain members: methods, fields, properties, events and nested types. If all of the packages are available (see [The Complete Profile](#)) in a particular implementation of the CLI then it is guaranteed that all of the following are also available:

- The base class for all classes in the package.
- All interfaces implemented by all classes and value types in the package.
- All types referenced in the members of all the classes in the package.

But with a factored library, these assumptions may not be valid for some implementations of the CLI. We will consider each of these assumptions below, but that discussion is simplified by the following requirements:

- All conforming implementations of the CLI shall implement the Kernel Package (see [The Kernel Package](#)).

- Any conforming implementation of the CLI that implements a package other than the Kernel Package shall implement the Basic Language Support Package (see [The Basic Language Support Package](#)).

#### 2.1.1 Missing Base Class

Interfaces do not have base classes. Value types have a base class of either **System.ValueType** or **System.Enum**, which are included in the Kernel Package, so they are always available.

For classes, however, it is possible that the base class is not available on the CLI implementation. In this case, the class itself will not be available as part of the package. That is, the existence of the class is dependent on the presence of *another* package. In the XML description of the BCL every such dependence is clearly described.

#### 2.1.2 Missing Interface

Interfaces, value types, and classes can all implement (or require the implementation of) an interface. But the dependence is less critical than a missing base class, since the interface does not contribute either shape or behavior (interfaces only specify requirements on the implementation).

Thus, if (for example) a class is defined to implement an interface that is not part of its own package, and the package that defines the interface isn't part of the implementation, it is simply as though the class hadn't been defined to implement the interface. Programmers who target such a class will not be able to cast instances to the interface, of course, because the interface itself isn't defined. In the XML description of the BCL every such dependence is clearly described.

#### 2.1.3 Missing Types for Members

When the type of a field is defined in a package that isn't present in the CLI implementation, the field isn't available. When any type required for a method (the return type and the types of all of its parameters) isn't present, then the method isn't available. The rule for methods applies for properties as well. Events do not have an associated type. In the XML description of the BCL every such dependence is clearly described.

### 2.2 Factoring and Profiles

The exact set of classes available in a package is *not* simply determined by the package, but rather by the total set of packages available in the CLI implementation. Similarly, the set members available in a class, value type, or interface is dependent on the set of packages available. The XML description of the BCL provides information about these dependencies.

This leads to these observations:

- The use of standard profiles dramatically simplifies the job of the application programmer. For a given standard profile, the list of classes, value types, interfaces, and members is fixed. This list can be precomputed and publicized for programmers targeting any given profile.
- Sometimes most of the classes in a package are unavailable unless another package is also present. These cases are specifically noted in the XML description of the BCL, and the definition of standard profiles takes these "practical dependencies" into account. When the percentage of classes with this kind of dependence becomes large the first package is listed as officially dependent upon the second package, and this is also called out in the XML description. If a conforming implementation of the CLI provides a package it shall also supply an implementation of all packages that the first depends upon.
- Sometimes most of the members of a class are unavailable unless another package is also present, making the class effectively unusable without the other package. These cases are specifically noted in the XML description of the BCL. If a conforming implementation of the CLI provides one of these packages but does not supply an implementation of the package required to make the class usable:
  - the CLI may omit the definition of the class
  - application programs shall not be written to assume that the classes are available unless it also requires the additional package

The standard profiles are defined to address this issue by specifying precisely what classes are available within that profile.

### **2.3 Factoring, File Format, and the Virtual Execution System**

The factoring of the Base Class Library is intended to allow simple and small implementations of not only the library itself but of the Execution Engine that implements the Virtual Execution System. While the file format used by the CLI is fixed independent of the particular packages that are supported, some parts of the file format are “Reserved for Other Use” unless an appropriate package is built into the CLI implementation. Similarly, while all CIL instructions are defined and their binary encodings are reserved, some of them need not be implemented by the EE unless a particular package is implemented. The precise relationship between implemented packages and parts of the File Format and CIL are described along with the package itself. See [The Kernel Package](#) for a complete description of features which are required only if packages are added to the minimal configuration.

### **2.4 Factoring and the Globalization Package**

For the most part, adding a package to an existing implementation of the CLI is simply a matter of defining the classes and methods specified in the package. In some cases, called out in the XML specification, it is also necessary to add classes or method to the other packages since the classes they depended upon are now available (see [Factoring: Categories, Packages, and Profiles](#)). In other cases, the underlying Execution Engine must be extended, as described with the package itself.

One package, however, has a more pervasive effect on the operation of the system. [The Globalization Package](#) adds the notion of both a Region and a Culture (commonly known collectively as a Locale) to the overall operation of the CLI. Without the Globalization Package, the Execution Engine and all of the base classes which deal with localizable objects (strings, dates and times, currency, etc.) are implemented for a single choice of locale and culture. In this case a conforming CLI implementation shall specify which locale and culture have been chosen by the implementation.

Addition of the Globalization Package, however, changes the behavior of all classes that can be localized. They shall then be dependent on ambient, programmatic, and administratively configured settings of the locale and culture. The programmatic interfaces are specified as part of the Base Class Library documentation. The ambient and administrative settings are outside the scope of the Standard but each conforming implementation of the CLI shall provide a description of whether and how these settings are implemented.

### **2.5 Factoring and Programming Languages**

A complete implementation of the C# programming language requires only features available from the Kernel, Basic Programming Languages, and Extended Numerics packages.

A complete implementation of the ECMAScript programming language requires only features available from the Kernel, Basic Programming Languages, Reflection , and Extended Numerics packages.

## **3 The Abstract Operating System Category**

This category contains a number of packages that provide an abstract, class-based interface to traditional operating system capabilities. These packages by themselves are not intended to encapsulate all of the features of a full operating system, but those that are in most common use by application programmers. Full access to the underlying operating system (if any) can be provided by a conforming CLI implementation by supplying [The Unmanaged Code Package](#) or by providing additional (non-Standard) class libraries.

### **3.1 The Networking Package**

This package provides access to networking capabilities, including:

- The equivalent of Sockets, for implementing connection-oriented or connectionless protocols
- Simple access to the TCP protocol for connection-oriented functionality
- Simple access to the UDP protocol for connection-less and multicast protocol functionality
- Simple access to the HTTP protocol for access to the World Wide Web

- An extensible framework for creating and using new or custom protocols
- Access to underlying network-based security services
- Miscellaneous common utilities, such as the manipulation of IP addresses, URIs, and access to the distributed name service.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package. The behavior of some of the classes in this package is impacted by whether or not The Globalization Package is also available, see Factoring and the Globalization Package.

#### **3.1.1 Impact on File Format, CIL, and Execution Engine**

None.

### **3.2 The Threading Package**

This package provides an implementation of threading (multiple execution contexts) within a single application domain. This is a complete implementation of a thread pooling model, including timers, call backs, and so forth. While it would be technically possible to implement this package without adding object locking to the underlying Execution Engine, the resulting system would be very difficult to use. Instead, we impose the requirement specified in 3.2.1.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### **3.2.1 Impact on File Format, CIL, and Execution Engine**

Any CLI compliant implementation that supports this package shall support object locking as specified, including special handling of synchronized methods and a means for locking individual objects.

### **3.3 The Security Package**

While the mechanism for defining a security policy is outside the scope of the Standard, it is expected that many implementations will use the extensibility model provided by **System.Security.Permissions** for that purpose. This package provides these major features:

- It provides the framework (**System.Security.Permissions**) in the Base Class Library required for programmers who wish to extend the built-in security model by defining new security permissions.
- It provides a basic set of security permissions that can be used as-is or specialized by programmers.
- It provides a secure isolated storage capability.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### **3.3.1 Impact on File Format, CIL, and Execution Engine**

None.

### **3.4 The Standard Input / Output Package**

This package provides a simple but powerful set of facilities for performing input and output. The major features are:

- Directories of files
- Files
- Streams
- Text handling facilities, including Unicode-related functionality

This package depends on the implementation of The Kernel Package and The Basic Language Support Package. The behavior of some of the classes in this package is impacted by whether or not The Globalization Package is also available, see Factoring and the Globalization Package.

### 3.4.1 Impact on File Format, CIL, and Execution Engine

None.

## 4 The Common Programming Utilities Category

This category specifies packages that provide general utilities to the end programmer. While all of this functionality could be written using a modern programming language without access to either the underlying operating system or the implementation of the CLI, it is provided here in a standardized form to promote code sharing and reuse.

### 4.1 The Collections Package

This package provides a set of interfaces that can be implemented by any class and are then used throughout the Base Class Library to provide common functionality. It also provides default implementations of many of these interfaces in the form of classes that can then be subclassed and specialized by programmers.

For example, the interface **System.Collections.IComparer** requires that the creator of a class that will implement **System.Collections.IComparer** provide a **Compare** method to allow items of their class to be compared to an arbitrary object and report back whether the two objects are equal to each other, the first less than the second, the second less than the first, or (via an exception) that they cannot be compared. The method **Sort** in the class **System.Array** (part of [The Kernel Package](#)) can then be applied to any array whose element type implements **System.Collections.IComparer**. The Collections Package also defines a class, **System.Collections.Comparer**, that provides a default implementation of this interface.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

#### 4.1.1 Impact on File Format, CIL, and Execution Engine

None.

### 4.2 The Common Data Types Package

This package provides a variety of general useful data types, including date/time, time spans, and mutable strings. It also provides general purpose formatting to and from strings, and conversion between numeric types.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#). The behavior of some of the classes in this package is impacted by whether or not [The Globalization Package](#) is also available, see [Factoring and the Globalization Package](#).

#### 4.2.1 Impact on File Format, CIL, and Execution Engine

None.

### 4.3 The Extended Numerics Package

This package supplies the user-visible class library support for floating point numbers, fixed point numbers (called **System.Decimal**), and a variety of mathematical operations that are not available in the minimal configuration of the CLI.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

#### 4.3.1 Impact on File Format, CIL, and Execution Engine

Any CLI compliant implementation that supports this package shall support floating point numbers. The behavior of all of the CIL instructions that deal with general numbers (e.g. **add**, **beq**, **ceq**, etc.) shall operate as specified on floating point values. Similarly, the general purpose instructions (**call**, **ldlema**, **ldfld**, **ldloc**, **pop**, **ret**, etc.) shall handle, as specified, floating point arguments. The following instructions shall be supported as specified: **ckfinite**, **conv.r.un**, **conv.r4**, **conf.r8**, **ldc.r4**, **ldc.r8**, **ldelem.r4**, **ldelem.r8**, **ldind.r4**, **ldind.r8**, **stind.r4**, **stind.r8**.

### 4.4 The Regular Expressions Package

This package provides functionality for doing sophisticated string matching operations.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 4.4.1 Impact on File Format, CIL, and Execution Engine

None.

#### 4.5 The Serialization Package

This package defines a user-extensible model for converting objects in memory into byte streams.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 4.5.1 Impact on File Format, CIL, and Execution Engine

None.

### 5 The Execution Engine Functionality Category

#### 5.1 The Garbage Collection Package

This package provides the **System.WeakReference** data type. This is the user-visible part of an optional mechanism that allows users to create references to garbage collected objects such that the object is permitted to be collected even though the reference is alive. See the class documentation for details.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 5.1.1 Impact on File Format, CIL, and Execution Engine

Any CLI compliant implementation that supports this package shall implement a garbage collector that supports weak references. Notice that finalization shall always be supported, since it is part of the core object model, exposed to users through a virtual method on **System.Object**.

#### 5.2 The Hosting Package

This package provides access to information about the application which is acting as host to the current invocation of the Execution Engine, as well as the underlying operating system and computer on which it is running.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 5.2.1 Impact on File Format, CIL, and Execution Engine

None.

#### 5.3 The Non-CLS Package

This package provides support for several features of the underlying Execution Engine that are not required for implementation of languages that only support the Common Language Subset (CLS).

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 5.3.1 Impact on File Format, CIL, and Execution Engine

Any CLI compliant implementation that supports this package shall provide the specified support for variable-length argument lists. This includes their metadata encoding in signatures, the ability to call them (affects the **call**, **calli**, **callvirt** and **jmp** instructions), and the ability to manipulate the argument lists (the **arglist** instruction shall be implemented).

Any CLI compliant implementation that supports this package shall also provide the specified for runtime typed pointers. The instructions **mkrefany**, **refanytype**, and **refanyval** shall be implemented.

Any CLI compliant implementation that supports this package shall also support structure layout attributes (sequential, explicit, packing, etc.).

## **5.4 The Reflection Package**

This package allows runtime access to the types that are currently available and their members. Using the Reflection Package a type name can be converted into an object that represents that type, and this can then be examined to find the names of methods, fields, etc. defined as part of that type. The Reflection Package also allows the values of fields to be read and written, and for methods to be called. All of this is under the control of a set of security permissions if [The Security Package](#) is also present.

The implementation of interpreters and dynamically typed languages (such as ECMAScript) often require the implementation of this package.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

### **5.4.1 Impact on File Format, CIL, and Execution Engine**

None.

## **5.5 The Unmanaged Code Package**

This package provides the ability to interoperate with code that was not designed to run within the Execution Engine itself. As such, it provides access to non-portable, platform-specific mechanisms. Use of this package can be expected to result in code that is not directly executable on other platforms or other conforming implementations of the CLI.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

### **5.5.1 Impact on File Format, CIL, and Execution Engine**

Any CLI compliant implementation that supports this package shall provide a garbage collector that supports pinning data. It shall also fully support P/Invoke metadata and the ability to call to unmanaged code and structure layout attributes (sequential, explicit, packing, etc.).

Note that this standard does not address the possibility of calling from unmanaged code into managed code. Whether and how this features is provided is left to individual implementations of the CLI.

## **6 The High Level Programming Category**

These packages provide functionality that is of interest for creating large scale applications. They deal with things such as XML encoding and transformation, asynchronous and remote programming, and globalization.

### **6.1 The XML Package**

This package provides functionality equivalent to the XML DOM specified by the World Wide Web Consortium. It provides both parsing and XML generation.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

#### **6.1.1 Impact on File Format, CIL, and Execution Engine**

None.

### **6.2 The Advanced XML Package**

This package provides an implementation of XML Query and XSL as specified by the World Wide Web Consortium.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

#### **6.2.1 Impact on File Format, CIL, and Execution Engine**

None.

### **6.3 The Asynchronous Programming Package**

This package provides support of asynchronous programming by defining the classes necessary to define asynchronous delegates and to wait for results being computed asynchronously.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 6.3.1 Impact on File Format, CIL, and Execution Engine

Any CLI compliant implementation that supports this package shall support asynchronous and one-way (“fire-and-forget” or “eventing”) behavior for delegates and other objects as specified.

### 6.4 The Globalization Package

This package provides the implementation of a general purpose localizable resource access mechanism, as well as the localized versions of certain resources (such as calendars). The impact of adding this package to an implementation of the CLI is more pervasive than simply adding the classes specified here. See Factoring and the Globalization Package.

This package depends on the implementation of The Kernel Package and The Basic Language Support Package.

#### 6.4.1 Impact on File Format, CIL, and Execution Engine

None.

### 6.5 The Remote Object Package

This package provides the support for a complete remote object package, including plug-in access to the underlying mechanism so that it can be extended by users to handle new protocols and data formats.

This package depends on the implementation of The Kernel Package, The Basic Language Support Package and The Serialization Package.

#### 6.5.1 Impact on File Format, CIL, and Execution Engine

Any CLI compliant implementation that supports this package shall support remote objects, including proxies and lifetime management services, as specified.

## 7 The Miscellaneous Category

### 7.1 The Kernel Package

This package is required of all conforming CLI implementations. It provides user access to the minimal features required of any Execution Engine.

#### 7.1.1 Impact on File Format, CIL, and Execution Engine

The following parts of the file format and CIL instructions need not be supported in a conforming implementation of the CLI that supports *only* the Kernel Package. The file format itself, and the binary encodings of the CIL instruction set, remain unchanged but the behavior of the Execution Engine is unspecified where these are used. A conforming program shall not make use of any of these features unless it has declared a dependence on a package that requires the feature, as listed here. These omissions constitute a minimal conforming implementation of the CLI.

- Asynchronous and one-way (“fire-and-forget” or “eventing”) behavior for delegates and other objects (required by The Asynchronous Programming Package).
- Floating point numbers (required by The Extended Numerics Package). The behavior of all of the CIL instructions that deal with general numbers (e.g. **add**, **beq**, **ceq**, etc.) need not consider the possibility of operating on floating point values. Similarly, the general purpose instructions (**call**, **ldelema**, **ldfld**, **ldloc**, **pop**, **ret**, etc.) need not worry about floating point arguments. The following instructions need not be supported at all: **ckfinite**, **conv.r.un**, **conv.r4**, **conf.r8**, **ldc.r4**, **ldc.r8**, **ldelem.r4**, **ldelem.r8**, **ldind.r4**, **ldind.r8**, **stind.r4**, **stind.r8**.
- Garbage collector support for weak references (required by The Garbage Collection Package). Notice that finalization shall always be supported, since it is part of the core object model, exposed to users through a virtual method on **System.Object**.
- Support for variable-length argument lists (required by The Non-CLS Package). This includes their metadata encoding in signatures, the ability to call them (affects the **call**, **calli**, **callvirt**

and **jmp** instructions), and the ability to manipulate the argument lists (the **arglist** instruction need not be implemented).

- Support for runtime typed pointers (required by [The Non-CLS Package](#)). The instructions **mkrefany**, **refanytype**, and **refanyval** need not be implemented.
- Remote object support, including proxies and lifetime management services (required by [The Remote Object Package](#)).
- Support for object locking (required by [The Threading Package](#)). If this package is not implemented there is no need to specially handle synchronized methods or to provide a means for locking individual objects.
- Garbage collector support for pinning data (required by [The Unmanaged Code Package](#)).
- P/Invoke metadata (required by [The Unmanaged Code Package](#)). Thus, the ability to call to unmanaged code or be called from unmanaged code is *not* required in minimal configurations.
- Support for structure layout attributes (sequential, explicit, packing, etc.) is required by both [The Unmanaged Code Package](#), and [The Non-CLS Package](#).

## 7.2 The Basic Language Support Package

This package adds to the minimum conforming implementation a set of packages that are required by most higher-level programming languages. Implementations of the CLI that do not provide this package are likely to be accessible only from assembly language or programming languages designed specifically for that implementation.

This package depends on the implementation of [The Kernel Package](#).

### 7.2.1 Impact on File Format, CIL, and Execution Engine

None.

## 7.3 The Development Time Package

This package contains two kinds of classes:

- A set of classes used to provide information to development-time tools. All references to these classes are intended to be removed before an assembly is loaded. For example, the class **System.Runtime.CompilerServices.AssemblyVersionAttribute** is used to inform a hypothetical linker tool to write a given version number into the final assembly that it produces. The linker is intended to remove the attribute when it creates the assembly.
- A set of classes which are expected to be provided in an implementation of the CLI for purposes of debugging an application, but which are not present in final product versions of the application. For example, **System.Diagnostics.BooleanSwitch** allows a program to test whether or not a Boolean flag used to control debugging output is set. When the application is shipped to customers it is intended that calls to this class will have been removed.

This package depends on the implementation of [The Kernel Package](#) and [The Basic Language Support Package](#).

### 7.3.1 Impact on File Format, CIL, and Execution Engine

None.

## 8 The Standard Profiles

The following profiles are composed of groups of the 22 packages described above and defined formally in the accompanying XML document. Each profile is self-consistent in the sense that if a package is included in the profile so are all the packages upon which it depends. A conforming implementation of the CLI shall implement one of these profiles.

**Rationale:** By standardizing a small number of profiles we provide clear guidance to applications programmers about what combinations of features they can expect to find commonly available. Without profiles there are over a million conforming implementations of the CLI (based simply on the choice of

implemented packages), making it difficult for an application programmer to target a significant percentage of the available CLI implementations. With a small number of profiles, each with a clear target application space, it becomes quite easy to target a large number of CLI implementations.

## **8.1 The Kernel Profile**

**Contents:** Kernel

This profile is the minimal possible conforming implementation of the CLI. It contains the set of classes that directly reflect a minimal implementation of the VES and the underlying mechanism for extensibility (custom attributes, for example).

## **8.2 The Compact Profile**

**Contents:** To be specified

This profile contains everything required to implement C#, the ECMAScript mobile profile proposed within ECMA TC39/TG1, and other functionality intended for use on resource-constrained embedded devices. The packages are chosen to allow implementation on devices with only modest amounts of physical memory.

## **8.3 The Complete Profile**

**Contents:** all packages

This profile contains all of the standardized packages and serves as the base for a desktop or server implementation of the CLI.

## **8.4 Others to be decided**





Free printed copies can be ordered from:

**ECMA**

114 Rue du Rhône  
CH-1204 Geneva  
Switzerland

Fax: +41 22 849.60.01

Email: [documents@ecma.ch](mailto:documents@ecma.ch)

Files of this Standard can be freely downloaded from the ECMA web site ([www.ecma.ch](http://www.ecma.ch)). This site gives full information on ECMA, ECMA activities, ECMA Standards and Technical Reports.

**ECMA**  
114 Rue du Rhône  
CH-1204 Geneva  
Switzerland

**See inside cover page for obtaining further soft or hard copies.**