Proposal for 1987 Smalltalk Standard primitives

prepared by:
L. Peter Deutsch, Xerox, ParcPlace Systems

This is a working document proposing the complete set of standard required primitives, and some standard optional primitives, for the 1987 version of the Smalltalk-80 language. Comments are solicited. Changes made in successive versions are not indicated in the text: consult the version history for a summary.

You will find this document easier to read if you print it in a fixed-width or nearly fixed-width font.

Version history:

[4] 25 February 1987: removed all primitives except those needed to support the language definition. This is the version distributed at the Feb. 26-27 implementors workshop

[3] 4 January 1987: changed standard primitives back to numbers; distinguished bitmaps from byte arrays. This is the version sent to Eliot Moss for distribution.

[2] 22 December 1986: added space inquiries; finished specifying the graphics primitives.

[1] 21 December 1986: first version.

The set of primitive methods is clearly one of the most implementation-dependent parts of Smalltalk-80 systems. Since our purpose in setting standards is to specify those things that all implementors can agree to support, we believe it is appropriate only to specify those primitive methods necessary to support the Smalltalk-80 language. In particular, we do not specify any primitives relating to I/O. We hope that a future effort will result in specifying standards in this area.

We recognize that implementors may choose to provide two different kinds of primitives in their systems:

- Primitives whose function is documented, but whose implementation is not accessible to users. We propose to specify these in terms of a standard class and selector, e.g. the primitive for adding SmallIntegers is called SmallInteger +. Some primitives of this kind are standardized here; implementors may add others at will. Compilers should give at least warning messages for primitives they do not recognize, and compile such methods as though the primitive specification were not there.
- Primitives defined by users, using an implementation-dependent

syntax. We propose to reserve a syntax for expressing these in the form of a string, but do not specify their semantics. For example, an implementation might allow users to write methods like

```
(SmallInteger) bitInvert
    <primitive: 'top ^= ~3;'>
    ^-1 - self
```

or

```
(SmallInteger) bitInvert
    <primitive: 'eorl #~3,top'>
    ^-1 - self
```

When porting an image or source code from one implementation to another, the default action should be to ignore primitives defined by strings.

For the remainder of this document, where we say "primitive" we mean "standard primitive", unless we explicitly say otherwise.

Even though the argument classes for primitives are often mentioned in the Blue Book, and are easily determined for those primitives for which the Blue Book supplies implementations, they are not documented in any one place, and there have been subtle disagreements between implementations, particularly on the issue of allowable receivers and arguments for the integer primitives. Consequently, we take this opportunity to specify the supported argument classes for all primitives. We also require the following properties of all (standard) primitives (and encourage, but do not require, implementors to observe them for implementation-specific primitives, as well as for standard ones):

- At the implementor's discretion, any primitive which is required to accept SmallIntegers as arguments is also allowed to accept LargeIntegers (positive, negative, or both, and perhaps only up to a certain size), and vice versa.
- If an integer result is expressible as a SmallInteger, it must be so expressed, even if it results from operations on LargeIntegers.
- At the implementor's discretion, any primitive which returns an integer result has the option of either only handling SmallInteger results (failing if the result cannot be represented as a SmallInteger), or of returning an appropriate LargeInteger in any particular case.
- At the implementor's discretion, any primitive which is required to accept Floats as arguments is also allowed to accept any subset of the Integer classes (small and/or large).

The current Smalltalk-80 VM does not provide any way for an implementor to check that an argument is a Semaphore, because Semaphore is not among the "well-known" classes listed on p. 576 of the Blue Book. This has caused considerable trouble in at least one implementation. We propose that ByteArray, Semaphore, and LargeNegativeInteger be recognized as classes in which a VM implementation may be interested, and allow implementors to take whatever steps are appropriate to make this possible.

We note that the implementor is NOT allowed discretion in the choice of acceptable receivers for a primitive, only in whether or not to allow a broader choice of arguments. In particular, we require that the compiler only allow a primitive to be attached to a method if the class in which the method is being compiled is among the acceptable receiver classes for that primitive: this may require a table lookup. We also

require that if a primitive accepts a given class as its receiver, it also accept all subclasses of that class.

Old primitives

Here is a complete list of the Blue Book primitives that we propose to retain as standard. For each primitive or range of primitives, we indicate its acceptable receiver class(es) (implicitly including all subclasses as well) and its acceptable argument class(es). Please refer to chapter 29 of the Blue Book, and especially the list of primitives on pp. 612-615, for additional detail on the list below.

Replaced

60-62 (Collection at:, at:put:, size)
These have been replaced by new, non-polymorphic primitives that are specific to the kind of collection being accessed: see below.

73, 74 (Object instVarAt:, instVarAt:put:)
A higher-level protocol, defined in the companion document on execution protocol, replaces these primitives.

75 (Object asOop)
This has been replaced by a primitive that returns a not necessarily unique hash value: see below.

87, 88 (Process resume, suspend)
See the companion document on execution protocol for the revisions of the process machinery.

Clarified

1-17 (SmallInteger arithmetic and comparison)
Receiver: SmallInteger; Argument: SmallInteger. As discussed above, these primitives may accept LargeIntegers as arguments, but they do NOT accept LargeIntegers as receiver: that is what primitives 21-37 do.

Changed

81 (BlockClosure value, value:*)
Receiver: BlockClosure, not BlockContext. See the companion documents on language definition and execution protocol for more information.

82 (BlockClosure valueWithArguments:)
Receiver: BlockClosure; Argument: Array. The argument must be an Array, not any other kind of collection.

Unchanged

40 (SmallInteger asFloat)
41-54 (Float arithmetic and comparison)
63-64 (String at:, at:put:)
70-71 (Behavior new, new:)
85-86 (Semaphore signal, wait)
110 (Object ==)
111 (Object class)

New primitives

Numbers

We propose to add the following new standard primitives for integers:
SmallInteger rem: SmallInteger -- there is no good reason to have //, \\, and quo: but not rem:.

Collections

We propose to change the primitives related to variable-size objects so that each one works only on a particular kind of object. Since we propose to eliminate objects containing both named and indexed instance variables from the language (see 'Classes' above), we propose that the primitives in the following list be installed locally in the class hierarchy where needed, and definitely not in class Object. The classes needing these primitives are probably:
ByteArray/Large(Positive/Negative)Integer/Bitmap for the byte-containing group, and Array for the pointer-containing group.

Array at: SmallInteger
Array at: SmallInteger put: Object
Array size
ByteArray at: SmallInteger
ByteArray at: SmallInteger put: SmallInteger
ByteArray size

The Smalltalk-80 system implicitly assumes that Strings and ByteArrays use the same stored representation. We hereby make this assumption explicit, at least to the extent of asserting that each element of a String occupies a single byte. However, to complete the semantic distinction between the two, we propose to add one new primitive for operating on Strings, namely:

String size

Miscellaneous

Object hash

This operation has the following properties:

- The primitive returns a non-negative SmallInteger.
- Distinct objects may have equal (primitive) hash values.
- An object's (primitive) hash value never changes.
- The (primitive) hash values of equal SmallIntegers are the same.

For the correct functioning of the Smalltalk system, classes that redefine = must also redefine hash so that any two equal objects (as determined by =) have the same hash.