

t Operations¹⁹

An operation is the basic executable component of a case: it does something. It has a name, zero or more inputs, zero or more outputs, and a distinctive icon. It can operate on input data and it can produce output data; it may also produce side effects (that is, beyond producing output data it may also change the state of an object, such as that of a window on the screen).

An operation can execute as soon as data has arrived on all its inputs.¹⁹

Cmd-clicking space in a case window creates an operation. A new, anonymous operation icon, with an insertion point within it for typing its name, appears. The type of operation, and consequently its icon, can be further specified by giving it one of many special names recognized by the Prograph system, or by transforming it through selecting appropriate menu items from the Opers or Controls menus.

Connectors¹¹⁰

Datalinks¹¹⁰

Data flows into an operation through terminals on the top side of the operation icon, and flows out through roots on the bottom side of the operation icon. Data flows from roots to terminals through datalinks.

A datalink between the root of an operation and a terminal of the same operation is not permitted (that would be a cycle). (However, see the “Loop” section below in this chapter). While a root can have multiple connecting datalinks, a terminal can have at most one connecting datalink.

The input bar and output bar of a case are two automatically supplied operations, which respectively bring input data into the case (by way of roots on the input bar) and export output data from the case (by way of terminals on the output bar). The input bar has no terminals, the output bar has no roots, and, like Locals, neither has a name.

Synchro Links¹¹⁰

A synchro enforces a sequential order of execution between operations. If the synchro is from operation A to operation B, it guarantees that B executes after A has executed. Note that this does not necessarily mean that B executes immediately after A; other operations can execute in between these two.

Names of Operations¹¹¹

In general, if an operation calls a user-defined method by name, the method to be applied is found according to the following conventions.

- o If the operation name is in the form `theName`, with no slash ("/") preceding the name, the operation calls a universal method. This is universal reference.

- o If the operation name is in the form `/theName`, the operation calls a method with name `theName` in the class of the instance that arrives as data on the leftmost input of the operation. This is data-determined reference. If the class is a subclass and does not have a method of its own called `theName`, an inherited method called `theName` is used. If no method called `theName` is found in the class or in any of its ancestors, a universal method called `theName` is used. If such a universal method is not found, a primitive called `theName` will be used, if it exists. Failing that, an error is signalled.

- o If the operation name is in the form `className/theName`, the method is sought in class `className` or, if it is not found there, in the ancestors of that class. This is explicit reference.

- o If the operation name is in the form `//theName`, the method `theName` is sought in the class of the case in which the operation resides (that is, in the class designated in the title bar of the case window containing the operation). This is context-determined reference.

Prograph Operations¹¹²

Operations come in a small number of different kinds, each with a distinctive icon.

The Prograph editor attempts to provide the default arity (number of inputs and outputs) for an operation when it is selected and the Return key is pressed. For class-based operations, a dialog appears asking the user to select from a list of classes with methods of that name (since methods of the same name in different classes can have different arity).

NOTE: Some primitives (system-supplied compiled operations), such as `+` and `pack`, have variable arity. In this case the arity displayed is the default arity. (See chapter 6, "Prograph Primitives.")¹¹³

Simple¹¹²

A Simple operation can call:

- o a primitive, of which there are two kinds.
 - a Prograph primitive (a system-supplied compiled operation) and
 -

an external primitive (XPrim); a user-provided compiled code resource—see appendix II, “C Code Usage”

a Macintosh Toolbox routine.¹²

a method (either universal or class-based), as determined by the naming conventions described above¹²

An operation when first created is Simple. After its name has been typed and the Return key pressed, Prograph parses the name and determines whether it refers to a primitive, a Toolbox call, or a user-defined method, altering the appearance of the operation icon accordingly. A primitive operation has a single line drawn on its bottom edge, a Mac Toolbox call has lines on both its top and bottom edges, and an operation that calls a user-defined method has no such lines.¹²

Constant¹²

A constant has a name and a single root. Its name is a textual representation of a simple value: a string, a list, or a number. When the operation executes, this value is made available on its root. The name of a constant is the displayable form of the value it produces.

Match¹³

A match operation has a name and a single terminal, and is always accompanied by a Control. The name, like that of a constant, is a textual representation of a simple value. When a match operation executes, its value is compared to the value flowing into its terminal. The match operation succeeds or fails with the success or failure of the comparison (See the “Controls” section of this chapter).

Persistent¹³

A persistent operation accesses the value of a persistent (see the “Persistents” section of this chapter). It has a name, at most one terminal, and at most one root. If it has an input, it is a Set Persistent operation: the value flowing into the input becomes the value of the persistent. If it has an output, it is a Get Persistent: the value of the persistent flows out its root.

Instance¹³

An instance operation has one input, one output, and a name. When executed, it produces a new instance of the class whose name it bears.

If the value of the input terminal is none (that is, it has no attached datalink), the instance is created with default attribute values as set in the class definition.

If the value of the input terminal is a list in the form ((attr-name-1 value-1) ... (attr-name-n value-n)), that is, if it is a list of attribute/value pairs, each pair taking the form of a list, then the instance is created with its attribute values set as specified by this input list.

If an Initialization method exists in the class specified by the name of the Instance operation, the instance is created, passed on as input to the Initialization method, and then appears on the root of the instance operation.¹³

Get¹⁴

A Get (short for “Get attribute”) operation accesses and returns the value of an attribute. It has a name (which is the name of the attribute to be accessed), one terminal, and two roots.

The input is either an instance of a class, or a string that is the name of a class. The latter option returns the default value of the attribute, and is also useful for accessing class attributes without first having to find or create an instance of the class.

The outputs are:¹⁴

- o On the left, the instance (if the input was an instance) or the string (if the input was the name of the class)
- o On the right, the value of the attribute

If the name of a Get operation is not preceded by a slash, “/” (that is, if it is a Universal reference), the value of the attribute is directly accessed.

If the name of a Get operation is preceded by a slash, “/” (that is, if it is a data-determined reference), the operation calls a Get method of that name in, or inherited by, the appropriate class. If such a Get method does not exist, the attribute is directly accessed.

Set⁻¹⁴

A Set (short for “Set attribute”) operation sets the value of an attribute. It has a name, which is the name of the attribute to be accessed, two terminals, and one root.

he inputs are:

- o On the left, an instance of a class, or a string that is the name of a class. If the input is a string the default value of the attribute will be set. This latter option is also useful for setting a class attribute without first having to find or create an instance of the class.
- o On the right, the value to which the attribute is set.

The output is either an instance (if the input was an instance) or the string (if the input was the name of the class).⁻¹⁴

If the name of a Set operation is not preceded by a slash, “/” (that is, if it is a Universal reference), the attribute is directly accessed and its value set.

If the name of a Set operation is preceded by a slash, “/” (that is, if it is a data-determined reference), the operation calls a Set method of that name in, or inherited by, the appropriate class. If such a Set method does not exist, the attribute is directly set.

Super⁻¹⁵

A super operation is a class-based operation with the search path for its applicable method set to start, not at its own class, but at the class of its immediate parent.

The name of a super operation is always in //theName format; that is, the class of the method called is determined by that of the method in which the operation appears. However, the search for the applicable method does not start in that class, but in the class immediately above it in the ancestral chain.

This is useful for writing a method in a subclass that shadows (has the same name as) a method of its parent class, but also uses that parent class’s method as part of its definition. Thus a method methodx can refer to its parent’s methodx without infinite recursion.⁻¹⁵

The Prograph editor permits applying Super to a selected operation by means of a menu item in the Controls menu. The operation’s icon receives an upward-pointing arrow on its right side, and the operation’s name is forced into //theName format.

Local⁻¹⁵

A Local (short for “Local Method”) is an encapsulation of a body of code into a single icon. It behaves very

much like a call to an unnamed method: like a method, it consists of a sequence of cases, but unlike a method, it is not called by name. A textual label (for example, get the name at left) can be attached to a local, but this serves purely as a descriptive comment.

A local is created by annotating an operation with Local from the Opers menu, or by selecting one or more operations and then selecting Opers to Local from the Opers menu. A Local icon appears with an insertion point within it for typing its textual label. Double-clicking the local opens a window onto its first case. If the local is created by means of Opers to Local, the selected operations appear within the first case window of the local.

If the local turns out to be useful enough to be upgraded to a method, callable by name from any part of the program, this can be done by means of the Local to Method menu item in the Opers menu. The user is asked to give the method a name. If this name is a universal reference (no slash), the method is a universal method. If the name is a data-determined reference (/theName), the user is asked to select a class to which the method can belong.

Evaluate

An Evaluate operation evaluates a simple mathematical expression. It has the following components:

Name

The name of the operation is the mathematical expression itself. Operators that can be used in the expression are listed below. All operators are binary (for example, $a \gg b$ right shifts a by b bits) unless indicated otherwise.

@

exponentiation

+

addition (binary and unary)

-

subtraction (binary and unary)

*

multiplication

/

division

//

integer division

%

remainder from integer division

&

bitwise AND

|
bitwise OR

^
bitwise XOR

~
bitwise NOT (unary)

<<
bit shift left

>>
bit shift right¹⁶

Operands (variables) can be the single letters a through z (upper and lower case are not distinguished). Parentheses can be used for precedence ordering. For a precise definition of the syntax and semantics of Evaluate see appendix IV.

Terminal*¹⁷

The input arity (number of terminals) of an Evaluate operation equals the largest position within the alphabet of the variables used in the expression.

That is, if the letter x is used as a variable, at least 24 terminals are created on the operation. If the letters a b d f are used as variables, six terminals are created, with the nth terminal corresponding to the nth letter of the alphabet. Thus the fourth terminal supplies a value for the variable d.

Root¹⁷

An Evaluate operation has one root, on which the result of the evaluation appears.

Macintosh Toolbox Operations¹⁷

Prograph supports calls to routines and accessors to data written in other languages. In particular, Prograph Classic supports such externals written in C or Pascal. Macintosh Toolbox trap calls, and accessors to Macintosh constants, globals, structures, and fields, are implemented in this way.

Mac Method¹⁷

Typing a name that is the name of a Macintosh Toolbox call into an operation and then pressing Return transforms that operation into a Mac Method call, with appropriate appearance (lines on top and bottom of the icon) and the correct number of terminals and roots.

Mac Constant, Mac Match, and Mac Global¹⁷

Macintosh Constants, Matches, and Globals are analogous to Prograph Constants, Matches, and Persistents, with the provision that their names must be valid names of Mac constants and globals.

Mac Get Field, Mac Set Field, and Mac Address¹¹⁸

Mac Gets and Sets are analogous to their Prograph counterparts, with the provision that their names must be valid names of Macintosh fields. They can also have an extra input to specify a zero-based index into an array.

A Mac Address operation takes a Mac structure as input, and returns a pointer to the field whose name it bears.

t Multiplex Operations¹¹⁸

A multiplex operation is an operation, either primitive or user-defined, that has been transformed into a repeating, parallel list-processing or looping construct.

The Prograph editor permits the following multiplex annotations:

- o Repeat can be applied to a selected operation.
- o List can be applied to selected terminals or roots of operations.
- o Loop can be applied to selected terminals or roots of operations.
- o Partition can be applied to a selected terminal of an operation with no roots.

List, Loop, and Partition terminals and roots have distinctive appearances, suggesting their respective functions. All multiplex operation icons are somewhat multilayered in appearance, suggesting that these operations execute repeatedly.

Repeat¹¹⁸

A Repeat operation executes repeatedly until a Fail, Terminate or Finish control within the called method is activated.

List¹¹⁹

The List annotation, when applied to the terminal of an operation, transforms that operation into one that applies itself to every element of the list that is expected on that input. A List-annotated root produces a list of results. The output list does not include any none values that may have been produced. An operation can have multiple list terminals and roots.

A list-annotated operation applies itself to each element of its input list or lists:

- o until every element of the shortest input list has been used, or
- o until a Fail, Terminate or Finish control within the called method is activated.

Loop⁻¹⁹

A Loop terminal always has a corresponding Loop root: data flowing out the Loop root is cycled back to its corresponding Loop terminal with each iteration of the operation until a Fail, Terminate or Finish control within the called Method is activated.

Partition⁻¹⁹

A Partition operation splits a list into two lists based on an arbitrary Boolean criterion. The Boolean criterion can be a Boolean primitive or a user-defined operation.

The Partition annotation is applied to an input terminal, transforming it into a List terminal on which the input list is expected to appear. Two distinctively shaped roots appear on the operation: the list of elements that satisfy the criterion flows out the left root, while the list of elements that fail the criterion flows out the right root.