

t

Controls^{-19"}

Flow of execution within a Prograph program is effected by Controls attached to operations. A control is activated on either success or failure of its operation. Prograph operations can succeed, fail, or be in error.

Success, Failure, and Error^{-20"}

Success^{-20"}

The default result for execution of an operation is success.

Failure^{-20"}

An operation fails if:

- o it is a match operation and the comparison it makes fails,
- o it is a Boolean operation with no roots that evaluates to FALSE (Note: a Boolean operation can have no root or one root. If it has one root, it either returns TRUE or FALSE. If it has no root, it either succeeds or fails.),
- o failure is propagated to it by an operation with a Fail control (see the "Fail" section below in this chapter).

Error^{-20"}

An operation generates an error if:

- o it is a primitive with inputs of inappropriate type or value,
- o it fails and does not have a control that fires on failure,
- o it has a Next Case control and the case in which the operation resides is the last case of the method,
- o the method called cannot be found,
- o the method found has arity different from that of the calling operation.

NOTE: If an error occurs, the interpreter beeps twice and the user is brought back to Editor context, with a case window open onto the offending operation, which is highlighted and blinking. An error message is also available.^{-20"}

The Controls^{-20"}

Prograph displays controls as small square icons attached to the right of operation icons. A control has two aspects: the action to be taken, and whether this action is taken on success or failure of the

operation. Control icons depict these two aspects as follows:

- o A check mark (√) within the control icon indicates that it is activated on success of the operation.
 - o An X within the control icon indicates that it is activated on failure of the operation (Note: We refer to either the √ or the X as an activation mark.)
 - o The other graphics within the square icon indicate the action to be taken.
- The controls and their actions are as follows.

Next Case

Abandon processing of the current case, and go on to execute the next case in the method. If there is no next case, signal an error.

The Next Case icon is plain, with nothing inside it except for its activation mark.

Continue

Continue execution of the current case. Continue on Success is the default for operations, and has no icon.

Continue on Failure has an X activation mark and miniature input and output bars within the control icon.

Terminate

Terminate execution of the current case immediately, and halt any repetition of the current method. If any values are output, these are the values computed on the last successful iteration of the method, or NULL if there was no iteration.

The Terminate control has a miniature input bar within its icon, to suggest that the values returned are those on the input bar (that is, those returned on the output bar on the last successful iteration).

Finish

Finish execution of the current case (that is, continue execution of the current case), and halt any repetition of the current method. The values output (if any) are those computed by the current case.

The Finish control has a miniature output bar within its icon.

Fail²²

Propagate failure to the calling operation.

At left, the write-line primitive returns 0 if no error occurs. The match-with-0 operation has a Fail on Failure control, so if write-line doesn't return 0, the match fails, and Failure is propagated to the write out operation, which in turn fails. There should be an appropriate control on the write out operation to handle that failure.

Inject²²

An Inject terminal supplies a name to an operation at runtime. A string flowing into the operation through the inject terminal becomes the operation's name.

Inject is applied to a selected terminal by means of the Inject menu item in the Controls menu. The terminal assumes a distinctive shape. Inject can be combined with List and Loop annotations.

In the example above, an instance is expected as input to the method. The attributes primitive returns, on its right root, a list of the names of the instance attributes of that instance. This list flows into the right terminal, annotated with both List and Inject, of the Get operation. The Get operation applies itself to every element of the incoming list, getting the value of the attribute named by that string element, and produces a list of attribute values on its list-annotated right root. ²²

t Persistents²³

Persistents are named elements that can hold any value. This value is retained between executions of a program and is also saved by the Prograph editor along with its program.

A Persistent is created by cmd-clicking space in the Persistents window. Its name is typed below its icon, and its value can be set at edit time by double-clicking its icon, which opens a Value window.

23

NOTE: When running in compiled mode, values of persistents can be saved and restored using the save and load primitives.
