7. MESSAGE SENDING FORMS

Objects in LOOPS communicate with each other by sending messages. This chapter describes the standard message sending forms used in LOOPS.

The following table shows the macros in this section.

Name	Type	Description
←	Macro and Function	Sends a message to an object.
SEND	Macro and Function	Sends a message to an object.
←**!**	Macro and Function	Evaluates the selector and sends a message to an object.
←**IV**	Macro	Invokes the function stored in an instance variable of the object.
←**Try**	Macro	Sends a message to an object only if it has a corresponding method.
←**Proto**	Macro	Sends a message to the prototype instance of a class.
←**Super**	Macro and Function	Combines an inherited method with local code; must appear in the body of a method.
←**Super?**	Macro	Combines an inherited method with local code; must appear in the body of a method. This does not cause an error if there is no inherited method.
←**SuperFringe**	Macro and Function	Invokes general methods for objects with more than one super class from which to inherit methods; must appear in the body of a method.
←**New**	NLambda NoSpread Macro	Creates an instance of a class and then sends a message to that instance.
FetchMethod	Macro	Finds the function name which implements the method invoked by a selector.

In addition, Chapter 8, Active Values, contains a description of ←**AV**, and Chapter 15, Performance Issues, contains a description of ←**Process** and ←**Process!**.

(← *self sel arg1 ... argn*) [Macro and Function]

Purpose: Sends the message with the selector *sel* to an object *self*. This is the standard way to send a message.

Behavior: Evaluates all arguments except *sel*.

When an object receives a message, it tries to match the selector *sel* with the names of its methods. If the object or the message does not recognize the message, a **Not Understood** error occurs.

The function version does more error checking than the macro and also attempts to convert unbound symbols into names for classes and instances.

Arguments:
- *self* — Pointer to an object.
- *sel* — Selector; not evaluated.
- *arg1...argn* — Arguments associated with *sel*.

Returns: The value returned by the method associated with *sel*.

Example: In this example, the message **New** is sent to the class **Window**. This returns the newly created instance.

```
76←(← ($ Window) New 'Window1)
#,($& Window (|OZW0.1Y:.;h.Qm:| . 495))
```

(**SEND** *self sel arg1 ... argn*) [Macro and Function]

Purpose: Same as ←, above.

Example: The expression

```
(SEND ($ Window) 'New 'Window1)
```

is equivalent to

```
(← ($ Window) New 'Window1)
```

(←**!** *self sel arg1 ... argn*) [Macro and Function]

Purpose/Behavior: Sends a message with the selector *sel* to an object *self*. It differs from ← in that it evaluates all of its arguments, including *sel.*

Arguments:
- *self* — Pointer to an object.
- *sel* — Selector, which is evaluated.
- *arg1...argn* — Arguments associated with *sel*.

Example: This example illustrates the fact that ←**!** evaluates the *sel* argument.

The code

```
(for sel in '(Shape Invert)
     do (←! ($ Window1) sel))
```

is equivalent to

```
(←Window1 Shape)(←Window1 Invert)
```

(←**IV** *self IVName arg1...argn*) [Macro]

Purpose: Invokes the function stored in the instance variable *IVName* of the object *self*.

Behavior: Gets a function from *IVName* of self and applies the function to *self* with the arguments *args*. Returns the value of the function or breaks.

←**IV** does not evaluate *IVName*.

Arguments:
- *self* — Pointer to an object.
- *IVName* — Instance variable name, which is not evaluated.

arg1...argn Arguments associated with *sel*; bound to arguments specified in the call.

(←**Try** *self sel arg1 ... argn*) [Macro]

Purpose: Sends the message with the selector *sel* to *self*, but only if there is a corresponding method.

Behavior: If *sel* is in fact a selector of *self*, the method is applied and the appropriate value is returned. If the method is not a selector of *self*, the symbol **NotSent** is returned.

Arguments:
self Pointer to an object.
sel Selector; not evaluated.
arg1...argn Arguments associated with *sel*.

Example: The expression `(←($ Window1) abcd)` normally causes a break.

```
79←(←Try ($ Window1) Update)
NIL

80←(←Try ($ Window1) abcd)
NotSent
```

(←**Proto** *class sel arg1 ... argn*) [Macro]

Purpose: Sends a message to the prototype instance of a class.

Behavior: Creates an instance of a class, if necessary, and puts that instance on the class variable **Prototype** of *class*, marking the class as changed. This instance is referred to as the prototype instance. **Proto** then sends the message *sel* to that instance.

Arguments:
class Pointer to a class.
sel Selector; not evaluated.
arg1...argn Arguments associated with *sel*.

Example: Usually only one instance of **LoopsIcon** is needed at a time, so the class **LoopsIcon** keeps one in its class variable **Prototype**.

```
81←(←Proto ($ LoopsIcon) Open)
```

(←**Super** *self sel arg1 ... argn*) [Macro and Function]

Purpose: Can invoke an inherited method within a method. ←**Super** must appear in the body of a method; it cannot be invoked directly.

Behavior: Searches up the class hierarchy and invokes the next more general method of the same name, even if a specialized method is inherited over a distance. It returns the value from that super method. You can use the form (←**Super**) when the arguments are not changed. If no arguments are provided, ←**Super** uses the arguments of the method from which it was called.

←**Super** and the other similar functions are now lexically scoped; that is, it is illegal o call ←**Super** anywhere but within a method body, and any selector given must be the same as the selector for that method.

Arguments:
self Pointer to an object.
sel Selector; not evaluated. Must be the same as the selector of the method in which the ←**Super** appears.

arg1...argn Arguments associated with *sel*.

Example: Two examples of ←**Super** are included:

- One example shows where the arguments are not changed.
- Oneexample shows where the arguments are changed.

Example 1: A use of ←**Super** where the arguments are not changed.

Define a subclass of **Window** that will call **RINGBELLS** before a window is shaped.

```
(DefineClass 'RingingWindow '(Window))
```

Through the browser interface, specialize the method **Shape**, to create the following method.

```
(RingingWindow.Shape
  (Method ((RingingWindow Shape)
           self newRegion noUpdateFlg)
  **COMMENT**    **COMMENT**
          (RINGBELLS)
          (←Super)))
```

Executing the following command calls **RINGBELLS** before the new window is shaped.

```
(←New ($ RingingWindow) Shape)
```

In the method above, if the positions of **RINGBELLS** and (←**Super**) were reversed, **RINGBELLS** would be called after the window was shaped.

Example 2: A use of ←**Super** where the arguments are changed.

Define a subclass of **Window** that will be square.

```
(DefineClass 'SquareWindow '(Window))
```

Through the browser interface, specialize the method **Shape**, to create the following method.

```
(SquareWindow.Shape
  (Method ((SquareWindow Shape)
           self newRegion noUpdateFlg)
  **COMMENT**    **COMMENT**
          (←Super self Shape
                       (create REGION
                          using
                           (SETQ newRegion
                             (OR newRegion
                                 (GETREGION)))
                           HEIGHT ←(fetch WIDTH
                                      of
                                    newRegion))
                      noUpdateFlg)))
```

Executing the following command creates a square window:

```
(←New ($ SquareWindow) Shape)
```

(←**Super?** *self sel arg1 ... argn*) [Macro]

Purpose: Invokes the single next most general method; must appear in the body of a method. This does not cause an error if no inherited method matches.

Behavior: Analogous to ←**Super**. The difference between ←**Super?** and ←**Super** is that ←**Super?** does not break if the *sel* does not have a more general method, whereas ←**Super** generates a break if there is not a more general method.

Arguments:

self	Pointer to an object.
sel	Selector; not evaluated. Must be the same as the selector of the method in which the ←**Super?** appears.
arg1...argn	Arguments associated with *sel*.

(←**SuperFringe** *self sel arg1 ... argn*) [Macro and Function]

Purpose: Invokes general methods for objects with more that one super class from which you wish to inherit methods; must appear in the body of a method.

Behavior: It invokes and executes the next more general method of the same name from each of the classes on the super's list *object's* class. Calling ←**SuperFringe** is analogous to sending ←**Super** up through each item on the super's list. If no arguments are provided ←**SuperFringe** uses the arguments of the method from which it was called.

Arguments:

self	Pointer to an object.
sel	Selector; not evaluated. Must be the same as the selector of the method in which the ←**SuperFringe** appears.
arg1...argn	Arguments associated with *sel*.

(←**New** *class selector arg1 ... argn*) [NLambda NoSpread Macro]

Purpose: Creates an instance of class and then sends *sel* and arguments to that instance.

Behavior: Creates a new instance of a class and sends a message to that instance. It returns the instance as a value and discards any value that may be returned by invoking the method specified by selector. ←**New** is equivalent to (← (← *ClassName* **New**) *selector arg1 ... argn*).

Arguments:

class	Pointer to a class.
sel	Selector; not evaluated.
arg1...argn	Arguments associated with *sel*.

Returns: The new instance.

Example: This example shows an example of ←**New** that creates a new instance of the class **Window** and asks you to shape it.

```
99← (←New ($ Window) Shape)
#,($& Window (|OZW0.1Y:.;h.Qm:| . 497))
```

(← *class* **FetchMethod** *sel*) [Method of Class]

Purpose: Finds the function name which implements the method invoked by sending a message with the selector *sel* to an instance of *class*. The function can be found in either *class* or its supers.

Behavior: Calls the function **FetchMethod**.

Arguments: *class* Pointer to a class.

sel Selector; evaluated.

Returns: The function for *sel* or NIL.

Example: Line 100 shows that the class **Window** implements the method **Update**.

```
100←(← ($ Window) FetchMethod 'Update)
Window.Update
```

Line 1 shows that neither the class **Window** nor any of its supers implements the method **abcd**.

```
1←(← ($ Window) FetchMethod 'abcd)
NIL
```

Line 2 shows that the class **Object** implements the method **PP** which will be triggered when instances of the class **Window** receive the **PP** message.

```
2←(← ($ Window) FetchMethod 'PP)
Object.PP
```

[This page intentionally left blank]