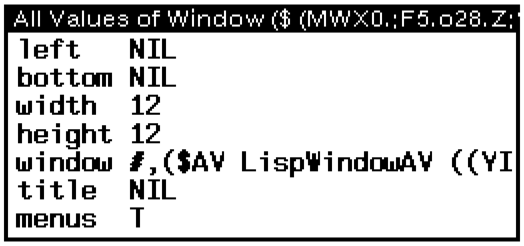
18. USER INPUT/OUTPUT MODULES

This chapter presents two modules that have been developed for displaying and allowing you to enter information. The Interlisp-D Inspector module and ?= handler have been enhanced to support LOOPS data types and message sending.

18.1 Inspector

The LOOPS interface uses and extends the capabilities of the Medley Inspector module. Instances and classes can be easily examined and modified through the interface that the Inspector module provides. This section describes the operations available with the LOOPS interface. For information on the Inspector module, see the *Interlisp-D Reference Manual.*

An inspector is a window opened on a specific piece of data, which for LOOPS means a class or an instance. Figure 18-1 shows an inspector on an instance of a window.



```
All Values of Window ($ (MWX0.;F5.o28.Z;
left   NIL
bottom NIL
width  12
height 12
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
```

Figure 18-1. Sample Inspector

Inspector windows contain two columns of information; the left column is called the property column, and the right column is called the value column.

You can scroll inspector windows, but these windows are not reshaped by actions such as switching from instance variables inspection to property inspection or adding new instance variables. This may cause some confusion, for example, if you create an inspector to be the correct size and add an instance variable, and that instance variable fails to appear.

An inspector is primarily an interactive facility. A programmatic interface is also available, which uses the LOOPS methods to customize the generic Interlisp-D functionality.

18.1.1 Overview of the User Interface

LOOPS provides two ways to create an inspector:

- Call the Lisp function **INSPECT** with the object to be inspected as the first argument.

- Use the LOOPS method **Inspect**, which is described below.

The user interface to the inspector is the same as that for the Medley environment; that is, you select an option from the left or right column with the left mouse button, and then trigger an action with the middle mouse button. The action opens a menu from which you can choose further options, like assigning a new value or adding an active value for breaking.

Another menu appears when you position the cursor on the title bar of an inspect window and press the middle mouse button. This menu allows you to change the inspector's contents, for instance to show all values or local ones for instance variables.

Three types of inspectors are available in LOOPS:

- Instance inspector
- Class inspector
- Class instance variable inspector

The following sections describe the user interface for each inspector.

(← *self* **Inspect** *INSPECTLOC*) [Method of Object]

Purpose: This provides a message form of the function to inspect the item *self.*

Behavior: Calls (INSPECT *self* NIL *INSPECTLOC*).

Arguments: *INSPECTLOC*
A region where the inspector window should appear. If it is NIL, you are prompted to place a ghost image.

Categories: Object

Example: The following command inspects an instance ($ W1).

```
17←(← ($ W1) Inspect)
```

18.1.2 Using Instance Inspectors

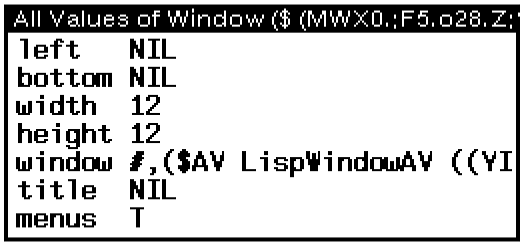
Using an inspector window on an instance provides a clear, direct interface to all of the instance's variables and values. This interface also provides the mouse and keyboard options to change the contents of the inspector to show various aspects of the instance.

18.1.2.1 Titles of Instance Inspector Windows

When you inspect an object, the title of the inspector window reflects the contents of the inspector. If the object is an instance, the title contains the name of the class of the instance and the LOOPS name of the instance, if it has one, or the UID of the instance. Other types of inspectors have different title bars, as described in Section 18.1.3, "Using Class Inspectors," and Section 18.1.4, "Using ClassIV Inspectors."

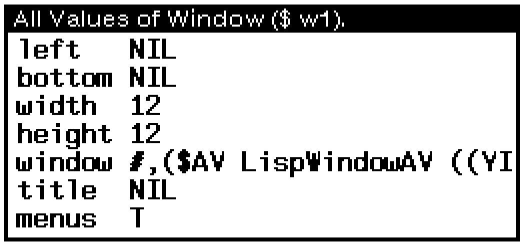
Contrast the title bar of the following two examples.

`(INSPECT (← ($ Window) New))` generates



```
All Values of Window ($ (MWX0.;F5.o28.Z;
left   NIL
bottom NIL
width  12
height 12
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
```

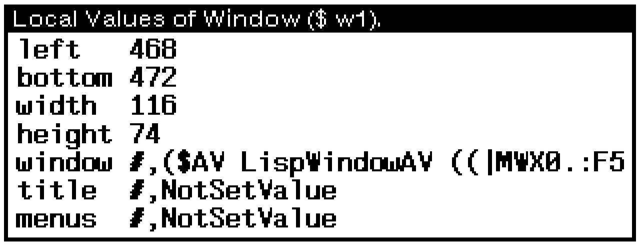
`(INSPECT (← ($ Window) New 'w1))` generates



```
All Values of Window ($ w1).
left   NIL
bottom NIL
width  12
height 12
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
```

18.1.2.2 Menu for the Title Bar

The following menu appears when you position the cursor on the title bar of the inspector window and press the middle mouse button.

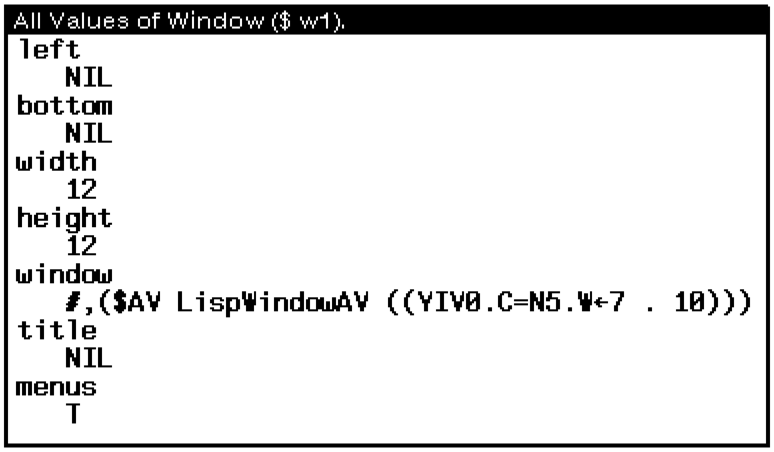


```
Local Values of Window ($ w1).
left   468
bottom 472
width  116
height 74
window #,($AV LispWindowAV ((|MWX0.:F5
title  #,NotSetValue
menus  #,NotSetValue
```

The rest of this section describes the actions that occur as a result of selecting one of the menu options.

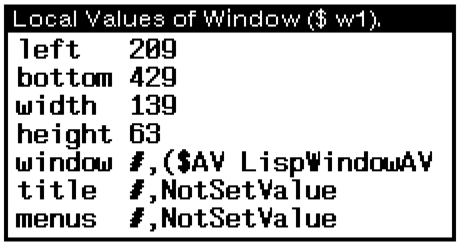
Class Opens a second inspector, a Class Inspector as described in Section 18.1.3, "Using Class Inspectors," which inspects the class of the instance within this inspector.

AllValues The default mode for instance inspectors. The values displayed in the right column of the inspector are determined by the function **GetValueOnly**, so active values (except #,NotSetValue) can be seen. The title of the inspector states that all values are being displayed.



```
All Values of Window ($ w1).
left
   NIL
bottom
   NIL
width
   12
height
   12
window
   #,($AV LispWindowAV ((YIV0.C=N5.W←7 . 10)))
title
   NIL
menus
   T
```

LocalValues The values displayed in the right column of the inspector are determined by the function **GetIVHere**. The title of the inspector states that only local values are being displayed. As with **AllValues**, active values are seen, and values that are not yet stored locally in the instance show a value of #,NotSetValue.



```
Local Values of Window ($ w1).
left   209
bottom 429
width  139
height 63
window #,($AV LispWindowAV
title  #,NotSetValue
menus  #,NotSetValue
```

Add/Delete Allows you to add or delete instance variables. Selecting this option pops up a new menu with two options:

- **Add**

  If you select **Add**, you are prompted to enter a name for the new instance variable. That instance variable is added locally to the instance and given the value of the variable **NotSetValue**. If you enter a name for an instance variable that currently exists, its value is reset to the value defined in the class.

- **Delete**

  If you select **Delete**, a menu appears with options that are the instance variables of the instance. If you select one that is not defined within the class, it is deleted. If the selected instance variable is defined by the class, a break occurs.

If the inspector is viewing the properties of an instance variable as opposed to all of the instance variables (see **IVs** below), the name entered under **Add** will be added as a property to that instance variable and given the value of the variable **NotSetValue**. If you try to delete an existing property, the menu that appears is a menu of property names.

IVs Changes the view to be one that shows all of the instance variables and their values, not the properties. It is possible to change the view an inspector has on an instance to show only the properties of a given instance variable. This is described in Section 18.1.2.3, "Using Commands in the Left Column," in the description of the Properties option.

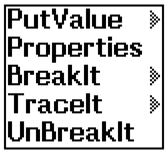
Save Value Calls **PutSavedValue** with its value argument bound to the instance being inspected.

Refetch Refreshes the inspector. Inspectors do not automatically update when a change is made to an instance, unless made with the **Edit** command.

Edit Opens a display editor window in which you can modify the value of instance variables and properties, and add or delete instance variables local to the instance.

18.1.2.3 Menu for the Left Column

The following menu appears when the view of the inspector is all of the instance variables of the instance, and you select an item in the left column and press the middle mouse button.



If the view of the inspector is only of properties, this menu contains only one option: **PutValue**.

The rest of this section describes the actions that occur as a result of selecting one of the menu options.

PutValue Allows you to assign a new value to the variable selected. Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **PutValue**

  Prompts you to enter a new value for this instance variable. The new value is stored using **PutValue**.

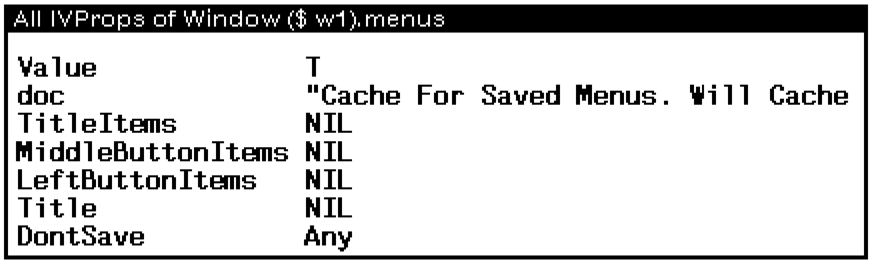
- **PutValueOnly**

  Prompts you to enter a new value for this instance variable. The new value is stored using **PutValueOnly**.

- **Use saved value**

  The new value to be stored using **PutValueOnly** is the value of **(SavedValue)** .

Properties Changes the view of the inspector to include only the value and properties of the selected instance variable as shown here:



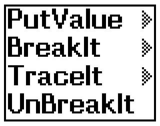
All IVProps of Window ($ w1).menus

```
Value             T
doc               "Cache For Saved Menus. Will Cache
TitleItems        NIL
MiddleButtonItems NIL
LeftButtonItems   NIL
Title             NIL
DontSave          Any
```

The title bar changes to indicate that the properties of an instance variable are being displayed, which instance is being displayed, and which instance variable of that instance is being displayed.

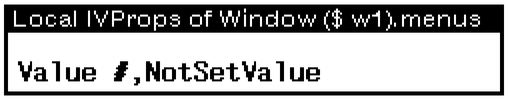
The Value item is provided purely as a convenience in this view and its menu options will only allow Putting a new value in it.

The following menu appears when the view of the inspector is an instance variable's properties, and you select an item in the left column and press the middle mouse button.



When the inspector's view is limited to IV properties the menu options act in a manner similar to that of IVs. This allows Putting, Breaking, Tracing and unBreaking of the properties instead of the IVs themselves.

If you now select the option **LocalValues** from the title bar menu, the title of the inspector changes to indicate that fact, and only properties that are stored locally in the instance appear, as shown here:



```
Local IVProps of Window ($ w1).menus

Value #,NotSetValue
```

To return to a view that shows all instance variables, choose the **IVs** option from the title bar menu.

BreakIt Wraps a **BreakOnPutOrGet** active value around the value of an instance variable. Any read or write accesses to this instance variable will cause a break (see Chapter 12, Breaking and Tracing).

Note: Breaking a variable effectively breaks any IndirectVariable that points to it.

Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **Break on Access**

  Performs the same action as **BreakIt**.

- **Break on Put**

  Installs a **BreakOnPut** active value. Trying to store a new value into this instance variable will cause a break, but reading the variable will not.

TraceIt Wraps a **TraceOnPutOrGet** active value around the value of this field. Any read or write accesses to this instance variable will be traced (see Chapter 12, Breaking and Tracing).

Note: Tracing a variable effectively traces any IndirectVariable that points to it.

Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **Trace on Access**

  Performs the same action as **TraceIt**.

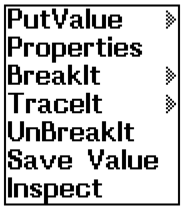
- **Trace on Put**

  Installs a **TraceOnPut** active value. All writes into this instance variable will be traced, but reads will not.

UnBreakIt Removes any of the breaks or traces that have been installed on an instance variable. If there are multiple traces or breaks, this will remove the outermost one.

18.1.2.4 Menu for the Right Column

The following menu appears when the view of the inspector is all of the instance variables of the instance, and you select an item in the right column and press the middle mouse button.



If the view of the inspector is only of properties of an instance variable, this menu contains only three options: **PutValue**, **Save Value**, and **Inspect**.

The only differences between these menus and the ones associated with the left column is the addition of two more options: **Save Value** and **Inspect**. The remaining options trigger identical behaviors as those of the menu associated with the left column.

Save Value Calls **PutSavedValue** with the selected value as its argument.

Inspect Calls the Lisp function **INSPECT** with the selected value as its argument, opening an additional inspector window.

In an inspector viewing the properties of an IV the right hand column middle button menu allows only the Put Value, SaveValue and Inspect options.

18.1.3 Using Class Inspectors

Classes can be inspected by using the Lisp **INSPECT** function or the LOOPS **Inspect** method (see Section 18.1.1, "Overview of the User Interface"). For example, to inspect the class **Window**, enter either of the following commands:

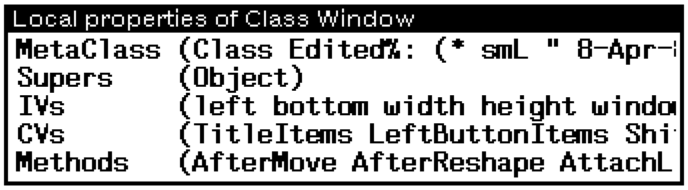
```
(INSPECT ($ Window))
(← ($ Window) Inspect)
```

The contents of a class cannot be changed from within an inspector window, so it is generally used for display as opposed to editing. However, the menu interface does provide ways to edit the contents of a class.

18.1.3.1 Titles of Class Inspector Windows

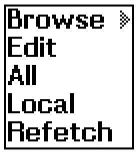
When you inspect a class, the title states that you are inspecting only local properties, and contains the name of the class.

The title contains the name of the class. The following example shows an inspector on the class **Window**. Since the value column is quite long, it has been truncated here.


```
Local properties of Class Window
MetaClass (Class Edited%: (* smL " 8-Apr-
Supers    (Object)
IVs       (left bottom width height windo
CVs       (TitleItems LeftButtonItems Shi
Methods   (AfterMove AfterReshape AttachL
```

18.1.3.2 Menu for the Title Bar

The title bar menu is associated with each inspector of a class. This menu appears when you position the cursor inside the title bar of the class inspector window and press the middle mouse button.



The rest of this section describes the actions that occur as a result of selecting one of the menu options.

Browse Provides a quick way to open a class browser on the class being inspected. Selecting this option and dragging the mouse to the right pops up a submenu with the following options:

- **Browse**

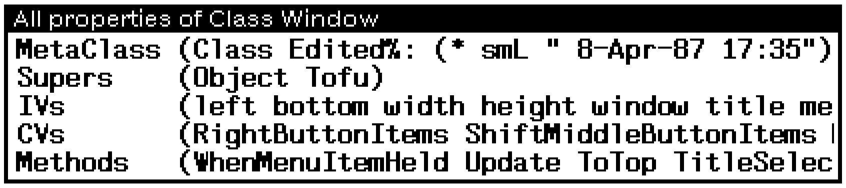
  Opens a class browser with the class being inspected as the root class.

- **BrowseSupers**

  Open a supers browser on the inspected class.

Edit Opens an editing window on the class.

All Causes the values shown in the right column to contain inherited as well as locally defined information. The title bar of the inspector changes to indicate this, as shown here:



```
All properties of Class Window
MetaClass (Class Edited%: (* smL " 8-Apr-87 17:35")
Supers    (Object Tofu)
IVs       (left bottom width height window title me
CVs       (RightButtonItems ShiftMiddleButtonItems |
Methods   (WhenMenuItemHeld Update ToTop TitleSelec
```

Local The default mode for class inspectors. This causes the values shown in the right column to contain only locally defined information, which is indicated in the title bar of the inspector.

Refetch Refreshes the inspector. Inspectors do not automatically update when a change is made to an instance, unless made with the **Edit** command.

18.1.3.3 Menu for the Left Column

No actions occur when you select an item in the left column of a class inspector and press the middle mouse button.

18.1.3.4 Menu for the Right Column

Only one option, **Inspect**, is in the menu that appears when you select an item in the right column of a class inspector and press the middle mouse button.

For the fields **MetaClass**, **Supers**, **CV**s, and **Methods**, selecting **Inspect** from the menu allows a choice of Interlisp-D inspectors. If the selected item in the class inspector is the values of the **IVs** field, then a ClassIVs inspector, described below, is created.

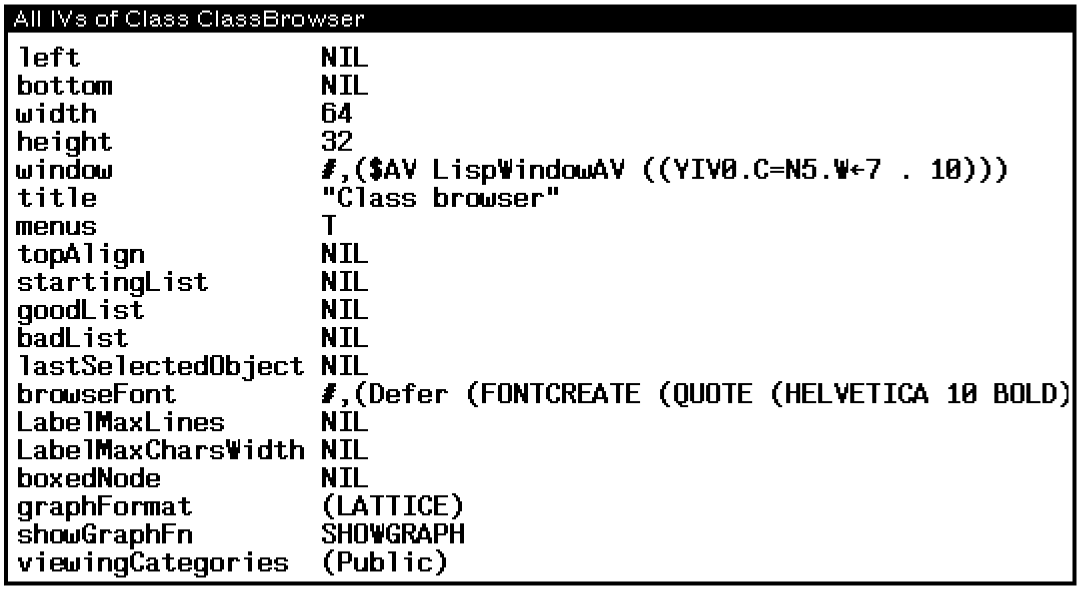
18.1.4 Using ClassIVs Inspectors

ClassIV inspectors provide an interface to the default values for all of the instance variables defined in a class. To create a ClassIV inspector,

- Open a class inspector.
- Select the values of the **IVs** field.
- Press the middle mouse button. This pops up and selects and **Inspect** menu, and automatically opens a ClassIVs inspector.

18.1.4.1 Titles of ClassIVs Inspector Windows

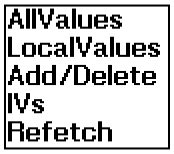
The title for a ClassIVs inspector indicates that the instance variables of a particular class are being inspected. This example shows how a ClassIVs inspector looks for the class **ClassBrowser**.



```
All IVs of Class ClassBrowser
left                NIL
bottom              NIL
width               64
height              32
window              #,($AV LispWindowAV ((YIV0.C=N5.W←7 . 10)))
title               "Class browser"
menus               T
topAlign            NIL
startingList        NIL
goodList            NIL
badList             NIL
lastSelectedObject  NIL
browseFont          #,(Defer (FONTCREATE (QUOTE (HELVETICA 10 BOLD)
LabelMaxLines       NIL
LabelMaxCharsWidth  NIL
boxedNode           NIL
graphFormat         (LATTICE)
showGraphFn         SHOWGRAPH
viewingCategories   (Public)
```

18.1.4.2 Menu for the Title Bar

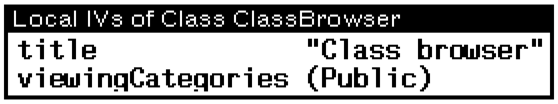
The following title bar menu is associated with each inspector of an instance. This menu appears when you position the cursor inside the title bar of the inspector window and press the middle mouse button.



The rest of this section describes the actions that occur as a result of selecting one of the menu options.

AllValues The default mode for ClassIV inspectors. Causes the inspector to show all instance variables, whether inherited or locally defined for the class, and states "AllIVs" in the title.

LocalValues Causes the inspector to show only locally defined instance variables. The following window shows how the title changes to indicate this:



```
Local IVs of Class ClassBrowser
title              "Class browser"
viewingCategories (Public)
```

Add/Delete Allows you to add or delete a ClassIV.

Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **Add**

If you select **Add**, you are prompted to enter a name for the new instance variable. That instance variable is added to the class and given the default value NIL and a **doc** property with the value (* IV added by (USERNAME)). If you enter a name for an instance variable that currently exists, its default value is reset to NIL.

- **Delete**

  If you select **Delete**, a menu appears with the locally defined instance variables of the class. Selecting one deletes it from the class.

If the inspector is viewing the properties of an instance variable as opposed to all of the instance variables (see **IVs** below), the name entered under **Add** is added as a property to that instance variable and given the value NIL. If you try to delete an existing property, the menu that appears is a menu of property names.

IVs Returns the view to show the instance variables and their values, not the properties.

It is possible to change the view a ClassIVs inspector has on the instance variables of a class to show only the properties of a given instance variable. This is described in Section 18.1.4.4, "Menu for the Right Column."

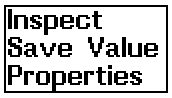
Refetch Refreshes the inspector.

18.1.4.3 Menu for the Left Column

No actions occur if you select an item in the left column of a ClassIVs inspector with the middle mouse button.

18.1.4.4 Menu for the Right Column

The following menu appears when you select an item in the right column and press the middle mouse button:

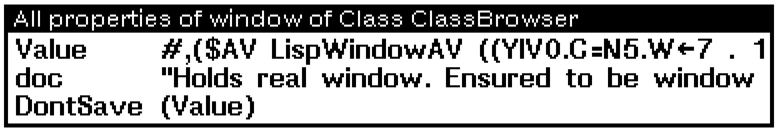


The rest of this section describes the actions that occur as a result of selecting one of the menu options.

Inspect Calls the Lisp function **INSPECT** with the selected value as its argument.

Save Value Calls **PutSavedValue** with the selected value as its argument.

Properties Changes the view of the inspector to display the value and properties of the selected instance variable, as shown in this example:



```
All properties of window of Class ClassBrowser
Value     #,($AV LispWindowAV ((YIV0.C=N5.W←7 . 1
doc       "Holds real window. Ensured to be window
DontSave (Value)
```

The title changes to include the following information:

- The properties of an instance variable.
- The name of the instance variable.
- The name of the class.

If you now select either **AllValues** or **LocalValues** from the title menu, the title of the inspector changes to indicate that fact, and the appropriate information is displayed.

18.1.5 Functional Interface for Instance Inspectors

The methods described in this section belong mostly to the classes **Class** or **Object**. Inspectors are not LOOPS objects, so these methods are invoked indirectly within the system functionality of the inspector as a customization of the Interlisp-D inspectors. These methods are meant to be called only from within the context that you create interactively by pressing a mouse button when the cursor is on some portion of an inspector window; you do not invoke them directly. Many of the parameters are simply passed along in case the method creates a menu, and the option selected from the menu needs additional arguments.

In these methods, the arguments *self* and *datum* may the same; that is, the item being inspected. The message is sent to the item being inspected, so its position in the inheritance lattice determines which method from the classes **Class** or **Object** is invoked.

The following table shows the items in this section.

Name	Type	Description
InspectFetch	Method	Returns the value of a left column inspector property that is displayed in the right column of an inspector window.
InspectStore	Method	Stores the value for an instance variable or its property.
InspectPropCommand	Method	After an item is selected in the left column of an inspector window, this triggers an action when the middle mouse button is pressed.
InspectProperties	Method	Determines what is displayed in the left column of an inspector window.
InspectTitle	Method	Creates a string to be used for an inspector window's title.
InspectValueCommand	Method	After an item is selected in the right column of an inspector window, this triggers an action when the middle button is pressed.
TitleCommand	Method	Triggers an action when the cursor is inside the title bar of the inspector window and the middle mouse button is held down.

(← *self* **InspectFetch** *datum property window*) [Method of Object]

Purpose/Behavior: Message sent by inspector to get the value of a left column inspector property that is displayed in the right column of an inspector window. Either **GetValueOnly** or **GetIVHere** is used to determine the value.

Arguments:

self The object being inspected.

datum This may or may not be a list. If it is not a list, it is bound to the object being inspected; that is, *self*. It is set to a list within various methods associated with the inspectors. The contents of this list are interpreted by a number of the methods to control what data is displayed within the inspector window.

- The first element of the list is the object being inspected.
- The second element of the list, if not NIL, is typically the name of an instance variable. In the terminology of the inspector, it is an inspector property. For inspectors of instances, the inspector properties (the items in the left column) are the instance variables of the object being inspected. (There can be some confusion here caused by using the word properties either when referring to the left column data of an inspector or when referring to the properties associated with an instance variable).
- The third element of the list, if NIL, indicates that inherited values are to be displayed in the inspector window; if its value is LocalValues, then only locally stored information is displayed.

The value of *datum* is stored on the inspector window property **DATUM**.

property Used if *datum* is not a list. Refers to an element (instance variable or property name) contained within the left column of an inspector. For an instance inspector, this could be either the name of an instance variable or the name of a property, depending upon the state of the inspector; that is, whether you are viewing instances variable or the properties of a particular variable.

window Lisp window of the inspector.

Returns: The value of a left column *property* that is displayed in the right column.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: The following command fetches the value of instance **W1'**s instance variable **bottom**:

```
15←(← ($ W1) InspectFetch (LIST ($ W1) 'window))
#,($AV LispWindowAV ((YIV0.C=N5.W←7 . 10)))
```

The following command fetches the value of class **Window**'s supers:

```
16←(← ($ Window) InspectFetch ($ Window) 'Supers)
(Object)
```

(← *self* **InspectStore** *datum property newValue window*) [Method of Object]

Purpose/Behavior: Stores *newValue* as the value for an instance variable or its property using **PutValueOnly**. Where the value is stored, whether in the instance variable or one of its properties, depends upon the values for *datum* and *property*.

Arguments:

datum	Instance or class being inspected. See **InspectFetch**, above, for details.
property	Instance variable or property where value is to be stored. See **InspectFetch**, above, for details.
newValue	New value for *property*.
window	Lisp window of the inspector.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: The following command changes the value of instance **W1**'s instance variable **height**:

```
17←(← ($ W1) InspectStore ($ W1) 'height 400)
400
```

or

```
18←(← ($ W1) InspectStore 'W1 'height 400)
400
```

(← *self* **InspectPropCommand** *datum property window*) [Method of Object]

Purpose: This method is an interface between LOOPS and the mouse functions of the inspector, and should only be called through the inspector. It is invoked when an item is selected in the left column of an inspector window and the middle mouse button is pressed.

Behavior: Opens a menu with a number of options. See Section 18.1.2.3, "Menu for the Left Column."

Arguments:

datum	Instance or class being inspected. See **InspectFetch**, above, for details.
property	Instance variable or property where value is to be stored. See **InspectFetch**, above, for details.
window	Lisp window of the inspector. A prompt window will be attached to this window if you ask to **PutValue**, and the window's **INSPECTW.FETCH** function will be called, so the window must be an inspector window.

Categories: Object

Specializations: Class

(← *self* **InspectProperties** *datum*) [Method of Object]

Purpose: Determines what should be displayed in the left column of an inspector.

Behavior: Depending on the value of *datum* as described above, this will return either the instance variables of the object being inspected, or the properties of a particular instance variable.

Arguments:	*datum*	Instance or class being inspected. See **InspectFetch**, above, for details.

Returns: Value depends on the arguments; see Behavior.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: The following command first shows the instance variables of instance **W1**, then the properties of the instance variable **height**:

```
27←(← ($ W1) InspectProperties 'W1)
(left bottom width height window title menus)

28←(← ($ W1) InspectProperties (LIST 'W1 'height))
(Value doc)
```

(← *self* **InspectTitle** *datum*) [Method of Object]

Purpose: Creates a string to be used as a title for an inspector window.

Behavior: If *datum* is not a list, this sets *datum* to (*datum* NIL NIL).

Depending on the values within the list *datum*, this creates a title showing whether all values or local values are shown and whether all instance variables or the properties of an instance variable are shown. The title also contains the LOOPS name or UID of *self*.

Arguments:	*datum*	Instance or class being inspected. See **InspectFetch**, above, for details.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: Some examples of the behavior of **InspectTitle**:

```
35←(← ($ Window) InspectTitle)
"Local properties of Class Window"

36←(← ($ W1) InspectTitle)
"All Values of Window ($ W1)."

37←(← ($ Window) InspectTitle (LIST 'Window T))
"All properties of Class Window"

38←(← ($ W1) InspectTitle (LIST 'W1 'height))
"All IVProps of Window ($ W1).height"
```

(← *self* **InspectValueCommand** *datum property value window*) [Method of Object]

Purpose: This method is an interface between LOOPS and the mouse functions of the inspector, and should only be called through the Inspector. It is invoked when an item is selected in the right column of an inspector window and the middle mouse button is pressed.

Behavior: Opens a menu with several options. See Section 18.1.2.4, "Menu for the Right Column."

Arguments:	*datum*	Instance or class being inspected. See **InspectFetch**, above, for details.
	property	Instance variable or property being inspected. See **InspectFetch**, above, for details.

	value	This is inspected only if you select **Inspect** from menu.
	window	Lisp window of the inspector.
Categories:	Object	
Specializations:	Class, InspectorClassIVs	
Example:	When the value of the instance variable **height** in instance **W1** is selected in an inspector window, and the middle mouse button is pressed, a message like the following is sent:	

```
(← ($ W1) InspectValueCommand ($ W1) 'height 200 (WHICHW))
```

(← *self* **TitleCommand** *datum window*) [Method of Object]

Purpose:	This method is an interface between LOOPS and the mouse functions of the inspector, and should only be called through the Inspector. It is invoked when the cursor is in the title bar of an inspector window and the middle mouse button is pressed.	
Behavior:	Brings up a menu with several options. See Section 18.1.2.2, "Menu for the Title Bar."	
Arguments:	*datum*	Instance or class being inspected. See **InspectFetch**, above, for details.
	window	Lisp window of the inspector.
Categories:	Object	
Specializations:	Class, InspectorClassIVs	
Example:	If you position the cursor inside the title bar of the inspector window for instance **W1** and press the middle mouse button, you send a message like the following:	

```
(← ($ W1) TitleCommand NIL (WHICHW))
```

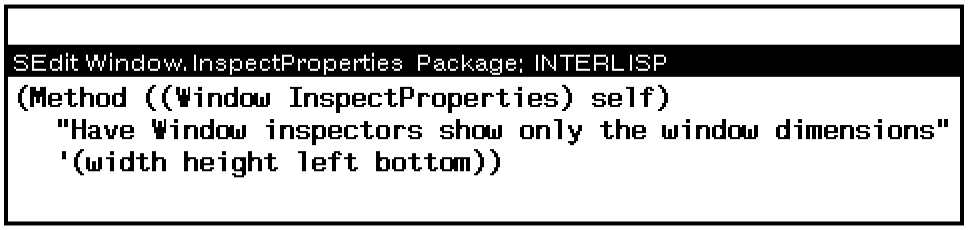
18.1.6 Customizing the Inspector

The methods in Section 18.1.5, "Functional Interface for Instance Inspectors," have been specialized in the classes **Class** and **InspectorClassIVs** to create the behavior of the inspectors described in Section 18.1.1, "Overview of the User Interface."

If you want to create a specialized inspector, you need to create a subclass of **Object** or perhaps **Class** and specialize the methods within that new class. The class **InspectorClassIVs** has an instance variable named **class** that contains the name of the class being inspected within a particular instance of **InspectorClassIVs**. Similarly, the user-created inspector class may need an instance variable which contains the object being inspected so that the methods of this class can easily access it.

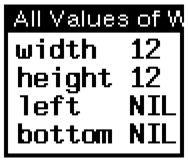
The methods that you need to specialize will depend upon how the behavior of the newly created inspector class should differ from those of an instance or class inspector.

As an example, assume that you want an inspector to show a subset of the instance variables of an instance. You could specialize the method **InspectProperties** to return that subset. To make **Window** show only the dimensions of a window, define the following method:



```
SEdit Window.InspectProperties  Package: INTERLISP
(Method ((Window InspectProperties) self)
   "Have Window inspectors show only the window dimensions"
   '(width height left bottom))
```

`(←New ($ Window) Inspect)` creates the following:



```
All Values of W
width  12
height 12
left   NIL
bottom NIL
```

18.2 Extensions to ?=

The Interlisp-D environment allows you to begin typing a function to be evaluated in the Executive and pause in the midst of typing the arguments. At this point, if you type a "?=" followed by a carriage return, Interlisp-D prints the arguments to the pending function call and shows the bindings. LOOPS has extended this facility to include similar functionality for message sending and for record creation.

18.2.1 Message Sending

The ?= interface works with the following message-sending forms:

- ←
- ←Super
- ←New
- ←Proto
- ←Process
- SEND

LOOPS first tries to determine the class of the object receiving the message by examining the form following one of the above. If the message form begins with one of ←**New** or ←**Proto**, the object receiving the message is the class desired.

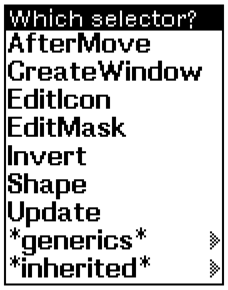
- If the system cannot determine the class of the object, you are prompted in the Prompt Window to enter in the name of the class or to type a right square bracket (]) to evaluate the form and determine that class from that. This handles cases such as

  ```
  (← (← ($ Window) New) ?=<CR>
  ```

- If the class can be determined and if you have not typed in a selector, a menu appears containing the options ***generics*** and ***inherited*** and any selectors local to the class. A submenu is associated with ***generics*** that contains selectors from the classes **Tofu**, **Object**, or **Class**, depending upon the class previously determined. The submenu associated with ***inherited*** are those selectors that are neither generic nor local to the class.

This is the menu that appears when you type

```
(←New ($ NonRectangularWindow) ?= <CR>
```



If you choose one of these options, it is placed into the input buffer and the system prints the binding for *self*, what method will be executed, and the arguments expected. In the prompt window, the system prints the documentation of the method to be executed.

As an example, if you create an instance of a class browser **cb1**, type

```
22←(← ($ cb1) ?= <CR>
```

and then choose **Shape** from the ***inherited*** drag-through menu, the Executive changes as shown here.

```
22←(←($ cb1) Shape
(←
self = ($ cb1)
Method = Window.Shape
 newRegion noUpdateFlg)
```

A similar output occurs if you type in a selector and "?=" instead of choosing a selector from the menu.

If you type "?=" after entering one or more of the arguments, the arguments are printed with the bindings, as shown here.

```
23← (←($ cb1) Shape '(100 150 200 250) ?= <CR>
(←
self = ($ cb1)
Method = Window.Shape
newRegion (QUOTE (100 150 ...))
noUpdateFlg)
```

This interface also works when you are typing to the edit buffer window when using the display editor. It does not work to pick a selector within a display editor window and choose the **?=** item from the **EditCom** submenu.

18.2.2 Record Creation

The same mechanism the LOOPS uses to handle ?= for LOOPS objects is also used to extend it for the Interlisp Record module.

If you begin an input to the Executive with one of **CREATE**, **Create**, or **create**, type the record name or data type to be created next, and then type

?=<CR>

the system prints the names but not the bindings of the fields within the record being created.

For example, when you type

```
48←(CREATE POSITION XCOORD ← 123 ?=<CR>
```

the response is

```
(XCOORD YCOORD)
```

on the next line. The caret moves to the position of the ?= in the original line, and waits for you to enter a value.

[This page intentionally left blank]