2. INSTANCES

Every object within the LOOPS system is an instance of some class. In this manual, however, the word instance generally refers to objects that are not themselves classes. Instances are a data type that contain local storage for instance variables, a pointer to the class that describes the instance, the Unique Identifier (UID), and other information.

This chapter describes naming and creating instances, accessing data stored within instances or pointed to by instances, and other related topics.

2.1 Instance Naming Conventions

A separate name space for LOOPS objects is maintained by the LOOPS system within a separate object name table. Since Lisp structures and LOOPS objects are stored in separate name tables, you can use the same symbol to refer to both a Lisp structure and a LOOPS object.

Note: The separate name space is not implemented by using the Common Lisp Package System.

Instances are not created with names; therefore, it may be necessary to keep pointers to them. Two ways are available to create pointers:

- Use Lisp variables, as in:

  ```
  (SETQ window1 (← ($ Window) New))
  ```

  This creates an instance of the class **Window** that can be referenced by the Lisp variable **window1**.

- Use a LOOPS name. This can be done in two ways:
  - Assign a name at the same time the instance is created. This can be done by using

    ```
    (← ($ Window) New 'window2)
    ```

    as described above. This creates an instance of the class **Window** that can be referenced by the LOOPS expression `($` window2).
  - Use the message **SetName** if you have a pointer to an object and want to assign a LOOPS name to that object.

The following table shows the items that manipulate LOOPS names.

Name	Type	Description
$	NLambda and Macro	Distinguishes between the Lisp value of a symbol and the LOOPS value of the same symbol; does not evaluate its argument.

$!	Function	Distinguishes between the Lisp value of a symbol and the LOOPS value of the same symbol; evaluates its argument.
SetName	Method	Assigns a LOOPS name to an object.
UnSetName	Method	Removes a name pointer to an object.
Rename	Method	Changes the name of an object.
GetObjectNames	Function	Returns the names of an object, including its UID.
ErrorOnNameConflict	Variable	Causes a break to occur when an attempting to name an object that already has a LOOPS name.

(**$** *name*) [NLambda and Macro]

Purpose/Behavior: Returns a pointer to a LOOPS object specified by the LOOPS name *name*. If no object exists for *name*, NIL is returned.

Arguments: *name* A LOOPS name.

Returns: Pointer to a LOOPS object or NIL; see Behavior.

Example: Given that

```
24←(← ($ Window) New 'window2)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))
```

then

```
25←($ window2)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))
```

The returned value is a pointer to the new window instance. For a further explanation, see Chapter 18, Reading and Printing.

(**$!** *name*) [Function]

Purpose/Behavior: Returns a pointer to an object specified by the value of the variable *name*, given that the value is a LOOPS name. If no object exists for *name*, NIL is returned.

Arguments: *name* Evaluates to a valid LOOPS name.

Returns: Pointer to a LOOPS object or NIL; see Behavior.

Example: Given that

```
26←(SETQ foo 'Window)
Window
```

and **Window** is a LOOPS object, then

```
27←($! foo)
#,($C Window)
```

(← *self* **SetName** *name*) [Method of Object]

Purpose: Assigns a LOOPS name to an object.

Behavior: If *name* is NIL, then a break occurs. If *name* is not a symbol, a break occurs. If *name* is already in use as a LOOPS name, and if the variable

ErrorOnNameConflict is non-NIL, then a break occurs, giving you the chance to OK "rebinding" *name*.

Note: If an object has multiple names, (← *self* **SetName** *NewName*) results in both the old name and new name appearing when (FILES?) is executed. The instance is also printed twice on the file if both names are specified to be saved.

Arguments: *self* An object.

name The LOOPS name to be given to the object; must be a symbol.

Returns: *self*

Categories: Object

Specializations: Class

Example: Given the commands

```
28←(SETQ window1 (← ($ Window) New))
#,($& Window (NEW0.1Y%:.;h.eN6 . 496))

29←(← window1 SetName 'window3)
#,($& Window (NEW0.1Y%:.;h.eN6 . 496))
```

the Lisp variable **window1** and the LOOPS expression

```
($ window3)
```

now point to the same object.

(← *self* **UnSetName** *name*) [Method of Object]

Purpose: Removes a LOOPS name pointer to an object.

Behavior: Removes the reference of *name* to *self* from the object name table maintained by the LOOPS system. If *name* is NIL, all names pointing to *self* in the object name table are removed from the files on **FILELST**. If *name* is non-NIL and the instance is associated with any files on **FILELST**, the instance is removed from those files. If *name* is not a valid LOOPS name for the object in question, an error occurs.

Arguments: *self* An object.

name A LOOPS name.

Returns: Used for side effect only.

Categories: Object

(← *self* **Rename** *newName oldNames*) [Method of Object]

Purpose: Changes the name of an object.

Behavior: If *oldNames* is NIL, removes all old names when *newName* is installed as the name for *self*; otherwise replaces only names specified in *oldNames* by *newName*. If *oldNames* is not a valid LOOPS name for the object in question, an error occurs.

Arguments: *self* Evaluates to a LOOPS name.

newName The LOOPS name to be given to the object; must be a symbol.

oldNames	List of symbols whose names are to be removed; if NIL, all old names are removed when *newName* is installed as the name for *self*.

Returns: *self*

Categories: Object

Specializations: Class

Example: Examine the following expressions to see the effects of **Rename**.

```
30←($ window2)
#,($& Window (NEW0.1Y%:.H53.G2A . 496))

31←(← ($ window2) Rename 'MyWindow)
#,($& Window (NEW0.1Y%:.H53.G2A . 496))

32←($ window2)
NIL

33←($ MyWindow)
#,($& Window (NEW0.1Y%:.H53.G2A . 496))
```

(**GetObjectNames** *object*) [Function]

Purpose/Behavior: Returns the names of *object*, including its UID.

Arguments:

object	A LOOPS object.

Returns: The names of *object*, including its UID.

Example: The command

```
(PROGN
  (← ($ Window) New 'w1)
  (← ($ w1) SetName 'w1again)
  (GetObjectNames ($ w1)))
```

returns

```
(w1again w1 (NEW0.1Y%:.H53.G2A . 497))
```

ErrorOnNameConflict [Variable]

Purpose/Behavior: Behavior depends on the value.

- If NIL, the existing object is replaced by a new object.
- If non-NIL, a break occurs when an attempt is made to give an object a name that is already in use as a LOOPS name.

Initially, the value for **ErrorOnNameConflict** is NIL.

2.2 CREATING INSTANCES

2.2 Creating Instances

When an instance is created by sending the **New** message to a class, the default behavior for **Class.New** is to send the message **NewInstance** to the newly created object. If you require that special or additional operations occur at instance creation time, specialize the method **NewInstance**. Specializations of the **NewInstance** method should return *self*. You also have the capability to pass arguments to the **NewInstance** method when the **New** message is sent to create the instance. For example, the following defines a class **NamedClass** which adds the instance variable **name** and specializes **New** to set that instance variable to the name of the instance when created.

```
(DefineClass 'NamedClass)
(←($ NamedClass) AddIV 'name)
(DefineMethod ($ NamedClass) 'New '(self name)
'(←@ (←self NewIstance name) name name))
```

You can also indicate whether instances are to be saved on files using the File Manager, which is described in Chapter 14, File Manager.

The following table shows the methods in this section.

Name	Type	Description
New	Method	Creates a new object of a particular class.
←**New**	Macro	Creates an object and sends a message to it.
NewInstance	Method	Allows initialization of newly created instances by class.
NewWithValues	Method	Creates an object with instance variables of assigned values.

(← *class* **New** *name arg1 arg2 ...*) [Method of Class]

Purpose: Creates a new object, which is an instance of the class *class*.

Behavior: Creates a new instance *name* and then sends the message (← "the new instance" **NewInstance** *name arg1arg2 ...*)

In the default case, the **New** method uses the default values for the instance variable values in the newly created instance. These default values are given in the instance variable descriptions of the given class. When that process is finished, the instance can be altered in various ways by sending it messages. Specializations of the **New** method should return the new instance, and can take more arguments after *name*.

The internal data structure of an instance contains a pointer to the class of which it is an instance.

Arguments:

class	Pointer to a class.
name	Name assigned to the instance; if NIL, object does not have a LOOPS name.
arg1arg2...	Arguments passed to the **NewInstance** method.

Returns: Newly created instance of the class.

Categories: Class

Specializations: AbstractClass, MetaClass

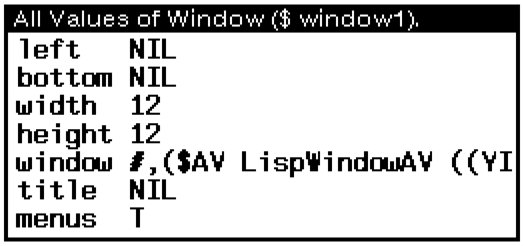
Example: The following command creates a new instance named **window1** of class **Window**.

```
20←(← ($ Window) New 'window1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 515))
```

The command

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

results in the following inspector window:



```
All Values of Window ($ window1).
left   NIL
bottom NIL
width  12
height 12
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
```

Some of the values assigned to the various instance variables are default values. These values are defined in the class **Window**.

(←**New** *class selector args*) [Macro]

Purpose: Creates an instance and sends a message to it within one form.

←**New** is pronounced "send new."

Behavior: Is equivalent to the form

(← (← *class* **New**) *selector args*)

Arguments:

class	Evaluates to a class.
selector	Name of the message to be sent to the new instance.
args	Arguments to be sent to the function invoked by the message.

Returns: The new instance.

Example: The command

```
23←(←New ($ Window) Open)
```

creates a new instance of the class **Window** and then sends the message **Open** to the newly created object.

(← *self* **NewInstance** *name arg1 arg2 arg3 arg4 arg5*) [Method of Object]

Purpose: Allows initialization of newly created instances by the class of the instance, as opposed to the metaclass. Subclasses of **Object** that specialize this method should have a ←**Super** form within the method to allow the execution of the default behavior.

Behavior: Not normally called directly, but is sent by method **New**. The default behavior is as follows.

If *name* is non-NIL, the message **SetName** is sent to *self*.

Within *self*, instance variables that are bound to the value of **NotSetValue** and have an **:initForm** property in the class description are filled. This allows you to override the **:initForm** behavior by setting values for instance variables before executing the ←**Super** form. See the discussion of **:initForm** in Section 2.3, "Data Storage in Instances at Creation Time."

Sends the message **SaveInstance** to *self* with the argument *name*.

Note: Specializations of the **NewInstance** method should return *self*.

Arguments:	*self*	Evaluates to a class.
	name	LOOPS name given to a new instance.
	arg1...arg5	Optional arguments referenced by user-written specialization code.

Returns: LOOPS name of new object created.

Categories: Object

Specializations: IndexedObject

(← *class* **NewWithValues** *valDescriptionList*) [Method of Class]

Purpose: Creates a new object and initializes the instance variables specified in *valDescriptionList*.

Behavior: Creates the object with no other initialization, directly installs the values and property lists specified in *valDescriptionList*, and returns the created object. Variables that have no description in *valDescriptionList* are given no value in the instance and thus inherit the default value from the class.

NewWithValues does not invoke the **NewInstance** method or the **:initForm** properties (see Section 2.3, "Data Storage in Instances at Creation Time"). This means that the instance is not recognized by the File Manager; to be recognized, the instance must be named.

Arguments: *class* Pointer to a class.

valDescriptionList
Evaluates to a list of value descriptions, each of which is a list of variableNames and properties, for example,

((VarName1 value1 prop1a propVal1a prop1b propVal1b ...)
(VarName2 value2 prop2a propVal2a prop2b propVal2b ...) ...)

Returns: The created object.

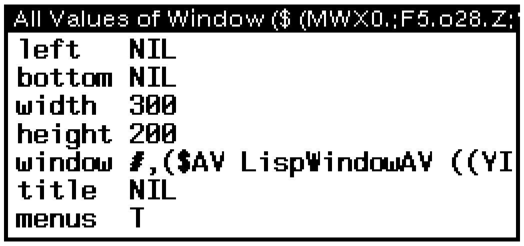
Categories: Class

Specializations: MetaClass

Example: The command

```
22←(INSPECT (← ($ Window) NewWithValues '((width 300)(height 200))))
```

results in the following inspector window:



```
All Values of Window ($ (MWX0.;F5.o28.Z;
left   NIL
bottom NIL
width  300
height 200
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
```

Contrast the values for the instance variables width and height with the inspector window for **New**, above.

2.3 DATA STORAGE IN INSTANCES AT CREATION TIME

2.3 Data Storage in Instances at Creation Time

When an instance is first created, the value of the variable **NotSetValue** is assigned to its instance variables. **NotSetValue** is initialized to be an active value of the class **NotSetValue** and should not be changed by the user. Trying to access an instance variable triggers this active value which in turn triggers the method **IVValueMissing**.

Data is stored in instances on all Puts and on **GetValues** when the default value is an active value but not **NotSetValue**. Be aware that in reading the value of an instance variable that is not stored in the instance, changes in the default value of the instance variable in the class description are seen in accesses of the instance.

One exception to this method of data storage at creation time is if an instance variable has the property **:initForm** in the class description. In this case, data is stored in the instance at the time of creation.

Testing for whether data is stored locally in the instance can be done in two ways:

- Through the user interface, you can inspect an instance in the local mode. (See Chapter 18, User Input/Output Modules, for more information.) Values not locally stored appear as #,NotSetValue.
- Programmatically, through the function **GetIVHere** with the macro **NotSetValue.**

The following table describes the items in this section.

Name	Type	Description
IVValueMissing	Method	Handles cases when an attempt is made to access the value of an instance variable that is not stored in an instance.
NotSetValue	Macro	Determines if its argument is equivalent to the value of **NotSetValue**.
:initForm	IV Property	Signals a property value that can be evaluated.

(← *self* **IVValueMissing** *varName propName typeFlg newValue*) [Method of Object]

Purpose: Invoked by the system to handle the cases when you try to access the value of an instance variable that is not stored in an instance. This is the mechanism the system uses to access default values.

Behavior: Varies according to the functionality that invoked it.

- **GetValueOnly** accesses return the default value of the instance variable stored in the class.
- **GetValue** accesses return the default value of the instance variable stored in the class if it is not an active value. If the default value is an active value, a copy of the active value is made, stored in the instance, and sent the **GetWrappedValue** message.
- **PutValueOnly** accesses store the new value in the instance.
- **PutValue** accesses store the new value in the instance unless the default value of the instance variable stored in the class is an active value. If this is the case, a copy of the active value is made, stored in the instance, and sent the **PutWrappedValue** message.

Arguments: *varName* Instance variable name.

propName	Property name for instance variable *varName*.
typeFlg	Used internally to indicate the type of access.
newValue	If called by **PutValueOnly** or **PutValue**, this is the value to be placed into the instance variable or property name.

Returns: Value depends on the functionality that invoked this method; see Behavior.

Categories: Object

(**NotSetValue** *arg*) [Macro]

Purpose: Determines if *arg* is **EQ** to the value of **NotSetValue**.

Arguments: *arg* Any value.

Returns: NIL or T.

Example: Given that

```
51←(← ($ Window) New 'w)
#,($& Window (NEW0.1Y%:.;h.eN6 . 515))
```

then

```
52←(NotSetValue (GetIVHere ($ w) 'title))
T
```

:initForm [IV property]

Purpose: This allows instance variables to be initialized at the time of the creation of an instance. The **:initForm** property and its value are in the class definition. Its value is a form that is evaluated when an instance is created. The result of the evaluation is stored as the value of the instance variable containing this property in the newly created instance.

This behavior does not hold if the value of the instance variable is not **NotSetValue**. Refer to the method **Object.NewInstance** in Section 2.2, "Creating Instances," for more information.

Example: Given the commands

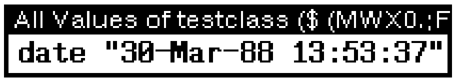
```
53←(DefineClass 'testclass)
#,($C testclass)


54←(AddCIV ($ testclass) 'date NIL '(|:initForm| (DATE)))
date
```

then

```
55←(INSPECT (← ($ testclass) New))
```

returns the following inspector window:


All Values of testclass ($ (MWX0.;F

```
date "30-Mar-88 13:53:37"
```

2.4 CHANGING THE NUMBER OF INSTANCE VARIABLES IN AN INSTANCE

2.4 CHANGING THE NUMBER OF INSTANCE VARIABLES IN AN INSTANCE

2.4 Changing the Number of Instance Variables in an Instance

An instance can contain more instance variables than are defined in the class that describes it. It is not possible to remove an instance variable from an instance if the instance variable is defined in the class.

When you try to access the value of an instance variable that is not defined as an instance variable in the instance, the **IVMissing** method is invoked.

The following table shows the functions and methods in this section.

Name	Type	Description
AddIV	Function	Adds an instance variable to an instance.
AddIV	Method	Adds an instance variable to *self*.
DeleteIV	Function	Removes an instance variable or property from an instance.
DeleteIV	Method	Removes an instance variable or property from *self*.
ConformToClass	Method	Makes *self* contain only those instance variables that are defined or inherited by the class of *self*.
IVMissing	Method	Is sent by the system when an attempt is made to access an instance variable that does not exist. It is used for recovery.

(**AddIV** *self name value propName*) [Function]

Purpose: Adds an instance variable to an instance.

Behavior: Varies according to the arguments.

- If *propName* is non-NIL and if *name* already exists, it is added as a property to the instance variable *name* with the value *value*.
- If *name* already exists, and if *propName* is NIL, the value of the instance variable *name* is changed to *value*.
- If *name* does not exist and if *propName* is non-NIL, the instance variable *name* is added to the instance and given the value of the variable **NotSetValue**. It is given the property *propName* with the value *value*.
- If *name* and *propName* already exist, the value of the property *prop* is changed to *value*.

Arguments:

self	A pointer to the instance.
name	The name of the instance variable to be added.
value	The value the new instance variable will be assigned.
propName	Property name of instance variable name; may be NIL.

Returns: Used for side effect only.

Example: Given that

```
55←(← ($ Window) New 'w)
```

the command

```
56←(AddIV ($ w) 'left 1234)
```

changes the value of the instance variable **left** to 1234. The command

```
57←(AddIV ($ w) 'foo 1234)
```

adds the instance variable **foo** to ($ w) and gives it the value 1234.

(← *self* **AddIV** *name value propName*) [Method of Object]

Purpose: Adds an instance variable to *self*.

Behavior: Method form of the function **AddIV**.

Arguments: See the function **AddIV**.

Returns: NIL

Categories: Object

Specializations: Class

Example: Given that

```
58←(← ($ Window) New 'w)
```

the command

```
59←(← ($ w) AddIV 'left 1234)
```

changes the value of the instance variable **left** to 1234. The command

```
60←(← ($ w) AddIV 'foo 1234)
```

adds the instance variable **foo** to ($ w) and gives it the value 1234.

(**DeleteIV** *self varName propName*) [Function]

Purpose: Removes an instance variable or property from an instance.

Behavior: Varies according to the arguments.

- If *self* does not have *varName*, an error occurs.
- If *varName* is defined in the class or a super class of *self*, an error occurs.
- If the instance *self* has *varName*, and *propName* is NIL, the instance variable is deleted.
- If *propName* is non-NIL, it is deleted only if it is a locally stored property, that is, not defined in a class. If *propName* is not a property of *varName* or is defined in a class, no error occurs.

Arguments:

self	A pointer to the instance from which the instance variable is to be deleted.
varName	The name of the instance variable to be deleted.
propName	If non-NIL, specifies that a property, not an instance variable, is to be deleted.

Returns: If no errors occur, this returns *self*.

Example: The following command deletes the instance variable **foo** from ($ w):

```
62←(DeleteIV ($ w) 'foo)
```

(← *self* **DeleteIV** *varName propName*) [Method of Object]

Purpose: Deletes an instance variable or property from *self*.

Behavior: Method version of the function **DeleteIV**.

Arguments: See the function **DeleteIV**.

Returns: If no errors occur, this returns *self*.

Categories: Object

(← *self* **ConformToClass**) [Method of Object]

Purpose/Behavior: Makes *self* contain only those instance variables that are defined in or inherited by the class of *self*.

Returns: NIL

Categories: Object

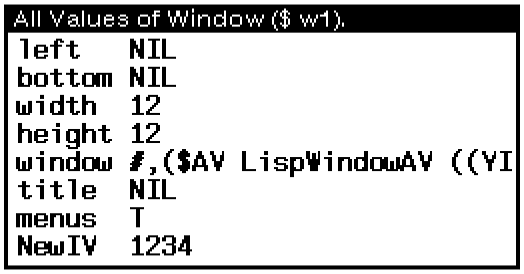
Example: This example adds an instance variable to an instance and shows how **ConformToClass** removes it.

```
63←(← ($ Window) New ’w1)
(#,($& Window (|MXWO.:F5.G18.Z:?|.18))

64←(← ($ w1) AddIV ’NewIV 1234)
1234

65←(INSPECT ($ w1))
```

This produces the following inspector window:

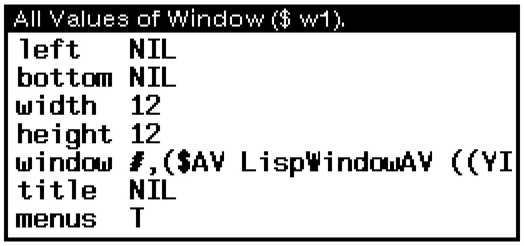


```
All Values of Window ($ w1).
left   NIL
bottom NIL
width  12
height 12
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
NewIV  1234
```

```
66←(← ($ w1) ConformToClass)
NIL

67←(INSPECT ($ w1))
```

This produces the following inspector window:



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

(← *self* **IVMissing** *varName propName typeFlg newValue*) [Method of Object]

Purpose: This message is sent by the system when an attempt is made to access an instance variable that does not exist. It is used for recovery.

Behavior: Varies according to the arguments.

- If the instance variable *varName* is now defined in the class, copy it to *self*. This can happen if the class was changed after the instance was created.
- If there is a class variable with the name *varName*, use it. The method of use is determined by the **:allocation** class variable property:
  - dynamicCached

    Copy the class variable to *self* on puts or gets.
  - dynamic

    Copy the class variable to *self* on puts. If the access is by **GetValue** or **GetValueOnly**, then get the value from the class. The value retrieved from the class is dependent on the value of *propName* and the class variable property **:initform**. If *propName* is NIL and there is a class variable property **:initform**, then retrieve the value returned from evaluating **:initform**. Otherwise, retrieve the value of the class variable *varName* if *propName* is NIL or the value of the property *propName* if it is non-NIL.
  - class (the default if there is no **:allocation** property)

    Do not copy the class variable *varName* to *self*. On puts, store the value in the class. With gets, do the same as the case when the **:allocation** property is dynamic. Essentially, this allows you to access class variables with the same syntax as instance variables.

  An attempt is made to correct the spelling of *varName* and try the above steps again before breaking.

Arguments:

self	A pointer to the instance.
varName	Instance variable name for *self*.
propName	Property name of instance variable *varName*.
typeFlg	One of **PutValue, PutValueOnly, GetValue, GetValueOnly**.
newValue	Value to be stored in *varName.*

Returns: If doing a put, this returns *NewValue*; else this returns the value of the instance variable name.

Categories: Object

Example: If **w1** is a **Window**, then the following command breaks under **Object.IVMissing** because windows do not have an instance variable named **mumble**.

```
(← ($ w1) Get 'mumble)
```

2.5 MOVING VARIABLES

2.5 Moving Variables

These functions allow you to move variables between classes.

Name	Type	Description
RenameVariable	Function	Changes a variable name in a class.

MoveVariable	Function	Moves an instance variable from one class to another.
MoveClassVariable	Function	Moves an class variable from one class to another.

(**RenameVariable** *className oldVarName newVarName classVarFlg*) [Function]

Purpose: Changes *oldVarName* to *newVarName* in class *className.*

Behavior: Can cause inconsistency without warning; does not test for references to the variable in methods of *className*.

Arguments:

className	Class in which function is defined.
oldVarName	Old name of variable.
newVarName	New name of variable.
classVarFlg	If not NIL, then *oldVarName* refers to a class variable.

Returns: If successful, returns *newVarName*; else NIL.

Example: The following command renames the class variable **OldVar** to **NewVar**.

```
27←(RenameVariable ($ MyClass) ’OldVar ’NewVar T)
```

(**MoveVariable** *oldClassName newClassName varName*) [Function]

Purpose: Moves an instance variable from *oldClassName* to *newClassName*.

Behavior: Moves both the *varName* instance variable and its description to *newClassName*. Deletes *varName* from *oldClassName*.

Arguments:

oldClassName	Source class.
newClassName	Destination class.
varName	Variable to be moved.

Returns: Used for side effect only.

(**MoveClassVariable** *oldClassName newClassName varName*) [Function]

Purpose: Moves a class variable from *oldClassName* to *newClassName*.

Behavior: Moves the class variable *varName* and its properties to *newClassName*. Deletes *varName* from the *oldClassName*.

Arguments:

oldClassName	Source class.
newClassName	Destination class.
varName	Class variable to be moved.

Returns: Used for side effect only.

2.6 DESTROYING INSTANCES

2.6 Destroying Instances

A protocol allows you to customize the behavior of the system at instance destruction time. The naming convention is somewhat asymmetrical to that of creation time. To programmatically influence instance creation, specialize the method **NewInstance**. To programmatically influence instance destruction, specialize the method **Destroy**. Include a (←**Super**) in specializations of **Destroy** to guarantee normal system behavior.

The following table describes the methods in this section.

Name	Type	Description
Destroy	Method	Removes an object from the environment.
Destroy!	Method	Removes an object from the environment. If the object is a class, it also destoys all subclasses.
DestroyInstance	Method	Modifies the data structure of an instance as described above.

(← *self* **Destroy**) [Method of Object]

Purpose: Removes an object from the environment.

Behavior: Sends the **DestroyInstance** message with *self* as an argument to the class of *self*. **UnmarkedAsChanged** is called to remove the instance from the notice of the File Manager.

Arguments: *self* A pointer to the instance.

Returns: Used for side effect only.

Categories: Object

Specializations: Class, DestroyedClass, IndexedObject, Window

Example: The following command destroys an instance named **window1**.

```
70←(← ($ window1) Destroy)
```

(← *self* **Destroy!**) [Method of Object]

Purpose/Behavior: Removes an object from the environment. If the object is a class, it also destoys all subclasses.

Arguments: *self* A pointer to the instance.

Returns: Used for side effect only.

Categories: Object

Specializations: Class, DestroyedClass, DestroyedObject

(← *class* **DestroyInstance** *instance*) [Method of Class]

Purpose/Behavior:	Destroys *instance* by overwriting its contents. When an instance is destroyed, several things occur:

- The instance is removed from any files on **FILELST**. See the *Interlisp-D Reference Manual*.
- The instance is deleted from system hash tables used for maintaining object identities.
- The class of the instance is changed to **DestroyedObject**.
- Other fields of the internal instance data structure are set to NIL.

If an instance is only pointed to by a LOOPS name, its data structure is freed for garbage collection.

Arguments:	*class*	Class of *instance.*
	instance	The instance being destroyed.

Returns: Used for side effect only.

Categories: Class

Specializations: MetaClass, DestroyedClass

2.7 Methods Concerning the Class of an Object

Given an instance, you often need to manipulate the class of an instance. This section describes how to perform this manipulation.

Name	Type	Description
ChangeClass	Method	Changes the class of an instance.
Class	Macro	Determines the class of an object.
Class	Method	Determines the class of an object.
ClassName	Function	Returns the class name of an object.
ClassName	Method	Returns the class name of an object.
InstOf	Method	Determines if *self* is an instance of a class.
InstOf!	Method	Determines if *self* is an instance of a class or any of its subclasses.

You can also compute a class corresponding to a Lisp data type for Lisp objects by using **GetLispClass**, described in Chapter 4, Metaclasses.

(← *self* **ChangeClass** *newClass*) [Method of Object]

Purpose: Changes the class of an instance.

Behavior: Creates a blank instance of the *newClass*. Any instance variables that are locally stored within *self* are added to the new instance.

If *newClass* is not the name of a class or a pointer to the class, an error occurs.

Arguments:	*self*	A pointer to an instance.
	newClass	Either the name of a class or a pointer to the class.

Returns: *self*

Categories: Object

Specializations: IndexedObject

Example: Create an instance of class **Window** and assign a local value to the instance variable **width** - all other instance variables of ($ w) lack local values. Then, when the class of ($ w) is changed to **IndirectVariable**, ($ w) will have all of the instance variables of its new class, plus the one instance variable of its old class which had a local value, **width**.

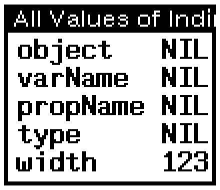
```
71←(← ($ Window) New ’w)
#,($& Window (NEW0.1Y%:.;h.eN6 . 501))

72←(←@ ($ w) width 123)
123

73←(← ($ w) ChangeClass ’IndirectVariable)
#,($& IndirectVariable (NEW0.1Y%:.;h.eN6 . 502))

74←(← ($ w) Inspect)
```

This produces the following inspector window:



```
All Values of Indi
object   NIL
varName  NIL
propName NIL
type     NIL
width    123
```

(**Class** *self*) [Macro]

Purpose: Determines the class of an object.

Behavior: If *self* is a LOOPS object, return its class.

If *self* is not a LOOPS object, evaluate (**GetLispClass** *self*)

Arguments: *self* A pointer to a LOOPS or Lisp object.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
75←(← ($ Window) New ’window1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 503))
```

then

```
76←(Class ($ window1))
#,($C Window)
```

Note: If *self* is an annotated value, the method **Class** and the macro **Class** return different values. See Chapter 8, Active Values, for more information on annotated values.

(← *self* **Class**) [Method of Object]

Purpose/Behavior: Method version of the macro **Class**.

Arguments: *self* A pointer to a LOOPS object or a Lisp data structure.

Returns: Value depends on the arguments; see Behavior of the macro **Class**.

Categories: Object

Example: Given that

```
77←(← ($ Window) New 'window1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 504))
```

then

```
78←(← ($ window1) Class)
#,($C Window)
```

(**ClassName** *self*) [Function]

Purpose: Returns the class name of the class of the object.

Behavior: Varies according to the argument.

- If *self* is a class, this returns the name of that class.
- If *self* is an instance, this returns the name of the class that describes that instance.
- If *self* is neither of these, an attempt is made to get the class of *self* by applying the function **GetLispClass** to *self*. If this returns NIL, the function **LoopsHelp** is called with the arguments *self* and "has no class name."

Arguments: *self* Can have multiple values; see Behavior.

Returns: Value depends on the argument; see Behavior.

Example: The command

```
80←(ClassName ($ Window))
```

returns

```
Window
```

(← *self* **ClassName**) [Method of Object]

Purpose/Behavior: Method version of the function **ClassName**.

Arguments: See the function **ClassName**.

Returns: Value depends on the arguments; see Behavior of the function **ClassName**.

Categories: Object

(← *self* **InstOf** *class*) [Method of Object]

Purpose/Behavior: Determines if *self* is an instance of *class*.

Arguments: *self* A pointer to an instance.

class A symbol name of a class or a pointer to a class.

Returns:	T or NIL
Categories:	Object
Example:	Given that

```
83←(← ($ Window) New 'w1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 505))
```

then

```
84←(← ($ w1) InstOf 'Window)
T

85←(← ($ w1) InstOf ($ Window))
T
```

(← *self* **InstOf!** *class*) [Method of Object]

Purpose:	Determines if *self* is an instance of *class* or any of *class*'s subclasses.	
Behavior:	Tests if class of *self* is a subclass of *class*.	
Arguments:	*self*	A pointer to an instance.
	class	A symbol name of a class or a pointer to a class.
Returns:	Object	
Categories:	Object	

2.8 Copying Instances

This section describes the methods for copying instances.

Name	Type	Description
CopyDeep	Method	Copies all nested objects, annotated values, and lists.
CopyShallow	Method	Creates a new instance of the same class as *oldInstance*. Fills the instance variables of the new instance with the data contained in the old instance.

(← *oldInstance* **CopyDeep** *newObjAList*) [Method of Object]

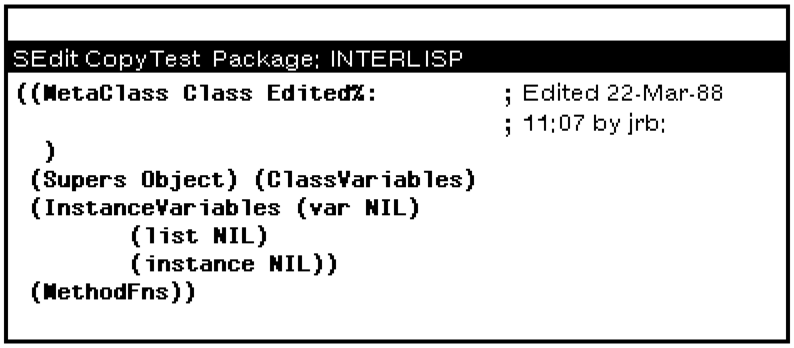
Purpose:	Copies all nested objects, annotated values, and lists. All other values are shared, not copied. This method is similar to the Interlisp function **COPYALL**.	
Behavior:	Creates a new instance of the same class as *oldInstance*. Fills the instance variables of the new instance with copies of lists, active values, and instances pointed to by *oldInstance*.	
Arguments:	*oldInstance*	A pointer to an instance.
	newObjAList	

An association list of old copied objects with their associated copies; used to allow copying of circular structures. Users typically let this argument default to NIL.

Returns: The value of the new instance.

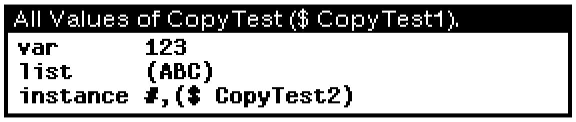
Categories: Object

Example: Create the class **CopyTest** with the following structure:



```
SEdit CopyTest  Package: INTERLISP
((MetaClass Class Edited%:          ; Edited 22-Mar-88
                                    ; 11:07 by jrb:
  )
 (Supers Object) (ClassVariables)
 (InstanceVariables (var NIL)
        (list NIL)
        (instance NIL))
 (MethodFns))
```

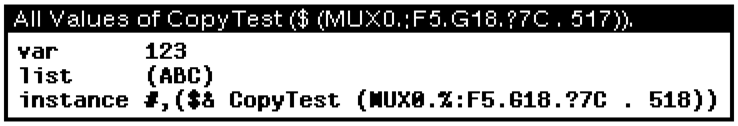
Create the instance **CopyTest1** and initialize it as shown in the following inspector:



```
All Values of CopyTest ($ CopyTest1).
var      123
list     (ABC)
instance #,($ CopyTest2)
```

Now create a copy and inspect it.

```
(INSPECT (SETQ DeepCopy (← ($ CopyTest1) CopyDeep)))
```



```
All Values of CopyTest ($ (MUX0.;F5.G18.?7C . 517)).
var      123
list     (ABC)
instance #,($& CopyTest (MUX0.%:F5.G18.?7C . 518))
```

The value of the instance variable **instance** is different. Also,

```
(EQ (@ ($ CopyTest1) list)(@ DeepCopy list))
```

returns NIL.

(← *oldInstance* **CopyShallow**) [Method of Object]

Purpose/Behavior: Creates a new instance of the same class as *oldInstance*. Fills the instance variables of the new instance with the data contained in the old instance.

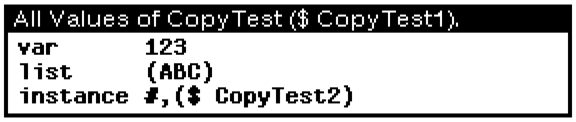
Arguments: *oldInstance*
A pointer to an instance.

Returns: A copy filled with the values shared by the instances.

Categories: Object

Example: Compare this example to the **CopyDeep** example above. Use the same **CopyTest1** instance as above.

```
(INSPECT (SETQ ShallowCopy (← ($ CopyTest1) CopyShallow)))
```



```
All Values of CopyTest ($ CopyTest1).
var       123
list      (ABC)
instance #,($ CopyTest2)
```

The value of the instance variable **instance** is the same. Also,

```
(EQ (@ ($ CopyTest1) list)(@ ShallowCopy list))
```

returns T.

2.9 QUERYING STRUCTURE OF INSTANCES

2.9 Querying Structure of Instances

At run time, user-written code may need to determine the structure of some object which has been passed into it. This section describes the methods to do this.

Name	Type	Description
HasCV	Method	Determines if a class variable can be accessed via *self*.
HasIV	Method	Determines if an instance variable can be accessed via *self*.
Inspect	Method	Inspects *self* as a class or instance.
ListAttribute	Method	Determines instance variable or instance variable property names contained in an instance.
ListAttribute!	Method	Recursively determines instance variable or instance variable property names contained in an instance.
WhereIs	Method	Searches the supers hierarchy to find a class where a specified name is defined.

(← *self* **HasCV** *cvName propName*) [Method of Object]

Purpose: Returns T if the class variable *cvName* (or its property *propName* if it is non-NIL) can be accessed via *self*; otherwise NIL.

Behavior: Sends the message **HasCV** to the class of *self* passing the arguments *cvName* and *propName*.

Arguments:

self	A pointer to an instance or a class.
cvName	Class variable name
propName	Property name for class variable *cvName*.

Returns: T or NIL; see Behavior.

Categories: Object

Specializations: Class

Example: The following command checks if an instance **window1** has the class variable **RightButtonItems**:

```
87←(← ($ window1) HasCV 'RightButtonItems)
T
```

(← *self* **HasIV** *ivName propName*) [Method of Object]

Purpose/Behavior: Returns T if the instance variable *ivName* (or its property *propName* if it is non-NIL) can be accessed via *self*; otherwise NIL.

Arguments:

self	A pointer to an instance or a class.
ivName	Instance variable name.
propName	Property name for instance variable *ivName*.

Returns: T or NIL; see Behavior.

Categories: Object

Specializations: Class

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

Purpose/Behavior: Inspects *self* as a class or an instance. Uses *INSPECTLOC* as the region for the inspector window if it is given.

Arguments:

self	A pointer to an instance.
INSPECTLOC	The region for the inspector window. If NIL, the system prompts you to place the window.

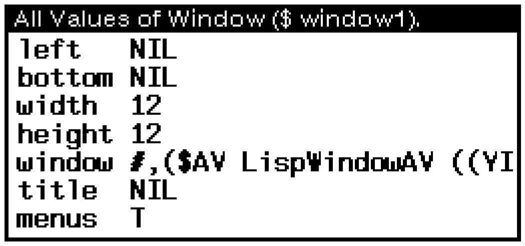
Returns: The Lisp window used by the inspector.

Categories: Object

Example: The following command inspects an instance ($ window1)

```
88←(← ($ window1) Inspect)
```

This results in the following inspector window:



```
All Values of Window ($ window1).
left   NIL
bottom NIL
width  12
height 12
window #,($AV LispWindowAV ((YI
title  NIL
menus  T
```

(← *self* **ListAttribute** *type name*) [Method of Object]

Purpose: Determines instance variable or instance variable property names contained in an instance.

Behavior: Converts *type* into uppercase on entry. The remaining behavior varies according to the arguments.

- If *type* is one of IV, IVPROPS, or NIL, and *name* is the name of an instance variable of *self*, this returns a list of property names of *name* that have property values locally stored in the instance.

- If *type* is IVS, this returns a list of the instance variable names of *self*, whether or not the values for the instance variables are locally stored.
- If *type* is none of the above, this evaluates (← (**Class** *self*) **ListAttribute** *type name*).

Note: Using a *type* of **SUPERS** or **SUPERCLASSES** returns a list of the names of the super classes.

Arguments:

self	A pointer to an instance.
type	See Behavior.
name	If *type* is one of IV, IVPROPS, or NIL, then *name* is an instance variable of *self*; else it is NIL.

Returns: Value depends on the arguments; see Behavior.

Categories: Object

Specializations: Class

Example: Given that

```
90←(← ($ Window) New 'w1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 515))
```

then

```
91←(← ($ w1) ListAttribute 'IVS)
(left bottom width height title menus)

92←(← ($ w1) ListAttribute 'IV 'menus)
NIL
```

After opening `($ w1)`, positioning the cursor anywhere on the window, and pressing the left and right mouse buttons to create some menus, then

```
93←(← ($ w1) ListAttribute 'IV 'menus)
(LeftButtonItems RightButtonItems)
```

(← *self* **ListAttribute!** *type name verboseFlg*) [Method of Object]

Purpose: Provides a recursive form of **ListAttribute**. Omits inheritance from the classes **Object** and **Tofu** unless *verboseFlg* is T.

Behavior: Converts *type* into uppercase on entry. The remaining behavior varies according to the arguments.

- If *type* is IVS, this is the same as **ListAttribute**.
- If *type* is one of IV, IVPROPS, or NIL, and *name* is the name of an instance variable of *self*, this returns a list of property names of *name*.
- If *type* is none of the above, this evaluates (← (**Class** *self*) **ListAttribute!** *type name*).

Note: Using a *type* of **SUPERS** or **SUPERCLASSES** returns a list of the names of the super classes.

Arguments:

self	A pointer to an instance.
type	See Behavior.
name	If type is one of IV, PROPS, or NIL, then *name* is an instance variable of *self*; else it is NIL.

verboseFlg	T or NIL; if T, inheritance from object and **Tofu** are included. If NIL, they are omitted.

Returns: Value depends on the arguments; see Behavior.

Categories: Object

Specializations: Class

Example: Given that

```
95←(← ($ Window) New ’w1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 515))
```

then

```
96←(← ($ w1) ListAttribute! ’IV ’menus)
(RightButtonItems doc TitleItems ...)
```

(← *self* **WhereIs** *name type propName*) [Method of Object]

Purpose: Searches supers hierarchy to find class where *name* is defined.

Behavior: Performs the method **Class.ListAttribute** for *self* and for each super class of *self*, checking to see if *name* (or *propName* as appropriate) is a member of the list returned. The value returned is the class where *name* (or *propName*) is first found.

The *type* argument is changed to uppercase and then coerced to a valued type argument for **ListAttribute**.

- If *type* is one of METHOD, METHODS, NIL, or T, it becomes METHODS. **WhereIs** then looks for a method with the name *name*.
- If *type* is one of IVPROP or IVPROPS, it becomes IVPROPS. **WhereIs** then looks for an instance variable property with the name *name*.
- If *type* is one of IV or IVS, it becomes IVS. **WhereIs** then looks for an instance variable with the name *name*.
- If *type* is one of CV or CVS, it becomes CVS. **WhereIs** then looks for a class variable with the name *name*.

Arguments:

self	A pointer to an instance.
type	See Behavior.
name	The name of an object attribute being searched for.
propName	Property name for instance variable *name*.

Returns: The class where *name* or *propName* is first found.

Categories: Object

Example: The command

```
97←(← (← ($ LatticeBrowser) New) WhereIs 'left 'IV)
```

returns

```
#,($C Window)
```

2.10 Other Instance Items

This section describes other items involved with instances.

NoValueFound [Variable]

Purpose/Behavior: Returned as a result of various accesses; initially set to NIL. When developing code, rebind this to the symbol **NoValueFound** to assist in debugging.

(**NoValueFound** *arg*) [Macro]

Purpose/Behavior: Returns value of (EQ **NoValueFound** *arg*).

Arguments: *arg* Any value.

Returns:	T or NIL.

(**ValueFound** *arg*) [Macro]

Purpose/Behavior:	Returns value of (NEQ **NoValueFound** *arg*).	
Arguments:	*arg*	Any value.
Returns:	T or NIL.	

[This page intentionally left blank]