5. ACCESSING DATA

This chapter discusses the various ways to access data:

- Generalized Get and Put functions
- Accessing data in instances
- Accessing data in classes

5.1 Generalized Get and Put Functions

These functions support generalized instance variable and property access for LOOPS objects. They can be very useful for implementing methods that support new types of conditional accessing; they have been used to simplify code in the active values system, for example.

This section deals with the following functions:

Name	Type	Description
GetIt	Function	Retrieves values from instance variables and properties.
GetItOnly	Function	Like **GetIt** , but returns active values on a variable/property without triggering them.
GetItHere	Function	Like **GetIt**, but returns active values on a variable/property without triggering them; does not observe **NotSetValue** as **GetItOnly** does.
PutIt	Function	Stores values into instance variables and properties.
PutItOnly	Function	Like **PutIt**, but stores by smashing active values on a variable/property without triggering them.

(**GetIt** *self varOrSelector propName type*) [Function]

Purpose: Retrieves values from instance variables and properties.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL
  - If *self* is an instance, this is equivalent to (**GetValue** *self varOrSelector propName*)
  - If *self* is a class, this is equivalent to (**GetClassIV** *self varOrSelector propName*)
- If *type* is 'CV,this is equivalent to (**GetClassValue** *self varOrSelector propName*)

- If *type* is 'CLASS,this is equivalent to (**GetClass** *self* (**OR** *varOrSelector propName*))
- If *type* is 'METHOD,this is equivalent to (**GetMethod** *self varOrSelector propName*)

Arguments:
self A class or an instance.
varOrSelector An instance variable name or the name of a method.
propName Property name.
type Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetIt ($ Window) 'doc NIL 'CLASS)
```

returns

```
"A Loops object that represents a window"
```

(**GetItOnly** *self varOrSelector propName type*) [Function]

Purpose: Retrieves values from instance variables and properties without triggering active values.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL
  - If *self* is an instance, this is equivalent to (**GetValueOnly** *self varOrSelector propName*)
  - If *self* is a class, this is equivalent to (**GetClassIV** *self varOrSelector propName*)
- If *type* is 'CV,this is equivalent to (**GetClassValueOnly** *self varOrSelector propName*)
- If *type* is 'CLASS,this is equivalent to (**GetClassOnly** *self* (**OR** *varOrSelector propName*))
- If *type* is 'METHOD,this is equivalent to (**GetMethodOnly** *self varOrSelector propName*)

Arguments:
self A class or an instance.
varOrSelector An instance variable name or the name of a method.
propName Property name.
type Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetItOnly (GetClassValue ($ LoopsIcon) 'Prototype) 'window)
```

returns the LoopsWindowAV that holds the image of the LOOPS icon. Calling **GetIt** with similar arguments returns the Lisp window object held by that LoopsWindowAV.

(**GetItHere** *self varOrSelector propName type*) [Function]

Purpose: Retrieves values from instance variables and properties without triggering active values; does not observe **NotSetValue** like **GetItOnly**.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL

- If *self* is an instance, this is equivalent to (**GetIVHere** *self varOrSelector propName*)

- If *self* is a class, this is equivalent to (**GetClassIVHere** *self varOrSelector propName*)

- If *type* is 'CV,this is equivalent to (**GetCVHere** *self varOrSelector propName*)

- If *type* is'CLASS,this is equivalent to (**GetClassHere** *self* (**OR** *varOrSelector propName*))

- If *type* is 'METHOD,this is equivalent to (**GetMethodHere** *self varOrSelector propName*)

Arguments:

self A class or an instance.

varOrSelector An instance variable name or the name of a method.

propName Property name.

type Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetItHere (GetClassValue ($ LoopsIcon) 'Prototype) 'title)
```

returns the value of **NotSetValue**. Calling **GetIt** with similar arguments returns the default value for this instance variable, NIL.

(**PutIt** *self varOrSelector newValue propName type*) [Function]

Purpose: Stores values into instance variables and properties.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL

- If *self* is an instance, this is equivalent to (**PutValue** *self varName newValue propName*)

- If *self* is a class, this is equivalent to (**PutClassIV** *self varName newValue propName*)

- If *type* is 'CV,this is equivalent to (**PutClassValue** *self varName newValue propName*)

- If *type* is 'CLASS,this is equivalent to (**PutClass** *self newValue* (**OR** *varName propName*))

Arguments:

self A class or an instance.

varName An instance variable name.

propName Property name.

type Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(PutIt (GetClassValue ($ LoopsIcon) 'Prototype) 'title "foo")
```

sets the instance variable **title** of the LOOPS icon prototype to "foo". This can be verified by inspecting (GetClassValue ($ LoopsIcon) 'Prototype) and examining the title slot.

(**PutItOnly** *self varOrSelector newValue propName type*) [Function]

Purpose: Stores values into instance variables and properties and smashes any active values it finds in its way without triggering them.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL
  - If *self* is an instance, this is equivalent to (**PutValueOnly** *self varName newValue propName*)
  - If *self* is a class, this is equivalent to (**PutClassIV** *self varName newValue propName*)
- If *type* is 'CV,this is equivalent to (**PutClassValueOnly** *self varName newValue propName*)
- If *type* is 'CLASS,this is equivalent to (**PutClassOnly** *self newValue* (**OR** *varName propName*))

Arguments: *self* A class or an instance.

varName An instance variable name.

propName Property name.

type Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: If the inspector from the **PutIt** example is used to set a break on the the instance variable **title** of the LOOPS icon prototype, then doing

```
(PutItOnly (GetClassValue ($ LoopsIcon) 'Prototype) 'title "mumble")
```

will set the instance variable **title** to "mumble" while smashing the trace active value.

5.2 ACCESSING DATA IN INSTANCES

5.2 Accessing Data in Instances

Two kinds of variables are associated with an instance:

- Its local instance variables, also referred to as IVs.
- The class variables, also referred to as CVs, that it shares with all instances of the class.

The data contained within instances are the values of instance variables and associated properties as well as a pointer to the class that describes the instance. Details of the LOOPS implementation determine exactly when the values of instance variables are stored within an instance. In some cases, the system must look to the class to find the values of instance variables. In general, you do not need to be concerned with this distinction; however, the details of it are covered in Chapter 2, Instances.

The types of data that an instance may contain is not limited. The values for an instance variable or a class variable can be any Lisp or LOOPS data structure.

The active value is a special case of data. When you try to access a variable with an active value as its value, the active value may be returned, depending upon the type of access. Normally, however, data computed by the active value is returned, not the active value. The details of how this computation is performed is described in Chapter 8, Active Values.

Instance variable names and class variable names are symbols and are not necessarily unique to each class. Although it is possible to use the same symbol for both a class variable name and an instance variable name, it is advisable not to do this since some of the LOOPS functionality examines both the instance variables and class variables in the search for data. See the method **IVMissing** in the class **Object**.

This section deals with the following functions and methods. See the *LOOPS Library Modules Manual* for information on how these interact with Masterscope.

Name	Type	Description
GetValue	Function	Finds the value of an instance variable.
Get	Method	Finds the value of an instance variable.
PutValue	Function	Writes the value of an instance variable.
Put	Method	Writes the value of an instance variable.
GetValueOnly	Function	Finds the value of an instance variable without triggering active values.
PutValueOnly	Function	Writes the value of an instance variable without triggering active values.
GetClassValue	Function	Returns the value of a class variable.
PutClassValue	Function	Changes the value of a class variable. The change occurs within the class and therefore causes all instances to access the new value of the variable.
GetClassValueOnly	Function	Returns the value of a class variable; does not trigger active values.
PutClassValueOnly	Function	Changes the value of a class variable. The change occurs within the class and therefore causes all instances to access the new value of the variable. Does not trigger active values.

GetIVHere	Function	Gets the value stored in an instance variable without invoking active values.

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

Purpose: Finds the value of an instance variable when *varName* and *propName* are to be computed.

Behavior: Varies according to the arguments.

- If *self* is an instance and *propName* is NIL, this returns the value of the instance variable *varName*. If there is no instance variable of the name *varName* and there is a class variable of that name, this returns the value of the class variable. See the **IVMissing** message for a complete discussion of this behavior. If there is neither an instance variable or class variable of that name, a break occurs.
- If *self* is an instance and *propName* is non-NIL, this returns the value of the property *propName* of the instance variable or class variable *varName*. If there is no property of the name, *propName*, this returns the value of the variable **NoValueFound**.
- If the value of *varName* (or *propName* if it is non-NIL) is an active value, the active value is activated.
- If *self* is not an instance, this calls `(`**GetIt** *self varName propName* `'IV)`

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments:

self	A class or an instance.
varName	Instance or class variable name.
propName	Property name.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
32←(← ($ window1) Shape '(100 200 300 400))
(100 200 300 400)
```

then

```
33←(GetValue ($ window1) 'width)
300

34←(GetValue ($ window1) 'LeftButtonItems)
((Update ...))
```

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

Purpose/Behavior: Method version of **GetValue**.

Arguments: See **GetValue**.

Categories: Object

(**PutValue** *self varName newValue propName*) [Function]

Purpose: Writes the value of an instance variable when *varName* and *propName* are to be computed.

Behavior: Varies according to the arguments.

- If *self* is an instance and *propName* is NIL, this changes the value of the instance variable *varName to newValue*. This returns *newValue*. If *varName* is not an instance variable of *self*, this causes a break.
- If *self* is an instance and *propName* is non-NIL, this changes the value of the property *propName* of the instance variable *varName* to *newValue*. If *propName* is not already a property of *varName*, it is added. This returns *newValue*.
- If the value of *varName* (or *propName* if it is non-NIL) is an active value, the active value is activated.
- If *self* is a class, this calls
  (**PutIt** *self varName newValue propName* `'IV)`

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments:

self	A class or an instance.
varName	Instance name or class name.
newValue	The new value for *varName* or *propName*.
propName	Property name.

Returns: Value depends on the arguments; see Behavior.

Example:
```
(PutValue  ($ window1) 'width 120)
```

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

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

Arguments: See **PutValue**.

Categories: Object

Specializations: Class

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

Purpose: Similar to **GetValue**, except that it overrides the active value mechanism.

Behavior: See **GetValue**. If the value found is an active value, it is returned without triggering its side effects.

Arguments: See **GetValue**.

Returns: See Behavior.

Example: The following expressions compare **GetValue** and **GetValueOnly**

```
35←(GetValue ($ window1) 'window)
{WINDOW}#nn,mmmm

36←(GetValueOnly ($ window1) 'window)
#,($AV LispWindowAV ...)
```

(**PutValueOnly** *self varName newValue propName*) [Function]

Purpose: Similar to **PutValue**, except that it overrides the active value mechanism.

Behavior: See **PutValue**. The argument *newValue* overwrites any active value on the slot without triggering it.

Arguments: See **PutValue**.

Returns: Used for side effect only.

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

Purpose: Returns the value of a class variable.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this returns the value of the class variable *varName*. If *varName* is not a class variable, a break occurs.
- If *propName* is non-NIL, this returns the value of the property, *prop*, of the class variable *varName*. If *varName* has no property of that name, the value of the variable **NoValueFound** is returned.

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments:

self An instance or a class.

varName Class variable name of *self*.

propName Property name for class variable *varName*; may be NIL.

Returns: Value depends on the arguments; see Behavior.

Example: The following commands show a variety of retuned values.

```
37←(GetClassValue ($ window1) 'window)
```

This breaks, since window is not a class variable of **Window**.

```
38←(GetClassValue ($ window1) 'LeftButtonItems)
 ((Update ...))

39←(GetClassValue ($ window1) 'LeftButtonItems 'qwerty)
NIL
```

(**PutClassValue** *self varName newValue propName*) [Function]

Purpose: Changes the value of a class variable. The change occurs within a class and therefore causes a class variable lookup by other instances to find the new value.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this changes the value of the class variable *varName* to *newValue*. If *varName* is not a class variable, this breaks.
- If *prop*Name is non-NIL, this changes the value of the property, *prop*Name, of the class variable *varName* to *newValue*. If *varName* has no property of that name, the property is added.

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments:

self An instance or a class.

varName Class variable name of *self*.

newValue Value to be assigned to class variable or property name.

propName Property name for class variable *varName*; may be NIL.

Returns: *newValue*

Example: The following command breaks since **left** is not a class variable name of **Window**.

```
40←(PutClassValue ($ window1) 'left 1234)
```

The command

```
41←(PutClassValue ($ window1) 'TitleItems 1234)
```

changes the value of **TitleItems**. The command

```
42←(PutClassValue ($ window1) 'TitleItems 123 'asdf)
```

adds the property **asdf** with the value 123 to **TitleItems**.

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

Purpose: Gets the value of a class variable without triggering active values.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this returns the value of the class variable *varName* without triggering active values. If *varName* is not a class variable, this breaks.
- If *propName* is non-NIL, this returns the value of the property, *propName*, of the class variable *varName* without triggering active values. If *varName* has no property of that name, the value of the variable **NotSetValue** is returned.

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments: *self* An instance or a class.

varName Class variable name for *self*.

propName Property name of class variable *varName*; may be NIL.

Returns: Value depends on the arguments; see Behavior.

Example: The following command returns the value of the variable **NotSetValue** since **LeftButtonItems** has no property of the name **qwerty**.

```
43←(GetClassValueOnly ($ window1) 'LeftButtonItems 'qwerty)
#,NotSetValue
```

(**PutClassValueOnly** *self varName newValue propName*) [Function]

Purpose: Changes the value of a class variable without triggering active values. The change occurs within a class and therefore causes a class variable lookup by other instances to find the new value.

Behavior: The behavior is the same as **PutClassValue** except that the value stored does not trigger an active value, but overwrites it instead.

Arguments: *self* An instance or a class.

varName Class variable name of *self*.

newValue Value to be assigned to class variable or property name.

propName Property name for class variable *varName*; may be NIL.

Returns: *newValue*

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

Purpose: Gets the value stored in an instance without invoking active values.

Behavior: Returns the value of *varName* (or the property, *propName*, if it is non-NIL) without triggering active values. If the value of *varName* (or *propName*) is not yet stored in *self*, the value of the variable **NotSetValue** is returned.

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments: *self* Must be an instance.

varName Instance variable of *self.*

propName Property name for variable *varName*; may be NIL.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
44←(← ($ Window) New 'w2)
#,($& Window (NEW0.1Y%:.;h.eN6 . 496))
```

then

```
45←(GetIVHere ($ w2) 'left)
#,NotSetValue
```

After entering the command

```
46←(PutValue ($ w2) 'left 123)
123
```

then

```
47←(GetIVHere ($ w2) 'left)
123
```

5.2.1 Compact Accessing Forms

When you write methods for classes that you have defined, there are a number of accesses to the data contained in the object bound to the method argument *self*. The following forms have been created to allow a more concise notation for these accesses.

Name	Type	Description
@	Macro	Provides compact **GetValue** and **GetClassValue** forms.
@*	Macro	Provides compact **GetValue** forms.
←@	Macro	Provides compact **PutValue** and **PutClassValue** forms and assigns a new value.

(@ *accessPath*) [Macro]

Purpose: Provides compact **GetValue** or **GetClassValue** forms.

Behavior: The *accessPath* can be one, two, or three arguments.

- If the *accessPath* is one argument, *self* is assumed to be the object and the argument points to an instance variable. This is the most common usage in methods in which you need to get the value of an instance variable contained in *self*. For example,

  ```
  (@ iv1)
  ```

  translates to

  ```
  (GetValue self 'iv1).
  ```

- If the *accessPath* is two arguments, the first argument is an object and the second argument is an instance variable. For example,

  ```
  (@ ($ w) left)
  ```

  translates to

  ```
  (GetValue ($ w) 'left).
  ```

- If the *accessPath* is three arguments, the first argument is an object, the second argument is an instance variable, and the third argument is a property. For example,

  ```
  (@ ($ w) menus DontSave)
  ```

  translates to

  ```
  (GetValue ($ w) 'menus 'DontSave).
  ```

When programming using objects, one object often points to another object. For example, the value of an instance variable is another object. Using different *accessPath* forms allows you to write accesses into objects that are nested inside of other objects. As an example, assume an object ($ pipe) has an instance variable named output with a value ($ tank), which has an instance variable named **level**. The command

```
(@ ($ pipe) output:level)
```

which is equivalent to

```
(@ (@ ($ pipe) output) level)
```

gets the value of the instance variable level of ($ tank).

The "**:**" is a delimiter that indicates instance variable access. The following table shows all the delimiters.

Delimiter	Description
:	Indicates instance variable access.
::	Accesses the value of a class variable whose name follows the double colon.
:,	Accesses the value of a property whose name follows the colon-comma.
.	Sends a message to the object with the selector following the period.
!	Evaluates the next expression.
\	States that the next symbol refers to a Lisp symbol. This is often used in conjunction with the exclamation mark, above.

$ States that the next expression is a LOOPS object.

You can test forms using these delimiters by evaluating `(Parse@ (LIST accessPath) 'IV)`.

Arguments: *accessPath* One, two, or three arguments; refer to Behavior.

Returns: See Behavior.

Example: The following examples show the (@ accessPath) form, the Parse@ test, and the translation.

1.
```
(@ foo)
(Parse@ (LIST 'foo) 'IV)
(GetValue self 'foo)
```

2.
```
(@ ::fie:foe)
(Parse@ (LIST '::fie:foe) 'IV)
(GetValue (GetClassValue self 'fie) 'foe)
```

The following three examples are rarely seen in code, but they are additional examples of the expressions that can be interpreted by the system.

3.
```
(@ foo::!::fum)
(Parse@ (LIST 'foo::!::fum) 'IV)
(GetClassValue (GetValue self 'foo)(GetClassValue self 'fum))
```

4.
```
(@ ($ w) fie:,foe.fum)
(Parse@ (LIST '($ w) 'fie:,foe.fum) 'IV)
(← (GetValue ($ w) 'fie 'foe) fum)
```

5.
```
(@ $fie.foe:!\fum.!foo)
(Parse@ (LIST '$fie.foe:!\fum.!foo) 'IV)
(←! (GetValue (← (GetObjectRec 'fie) foe) fum)(GetValue self 'foo))
```

(@* *accessPath*) [Macro]

Purpose/Behavior: Provides a concise form for writing embedded **GetValue** forms.

Arguments: *accessPath* An object followed by an arbitrary number of instance variable names.

Returns: The value of a nested instance variable.

Example: The command

```
(@* ($ foo) a b c)
```

translates to

```
(GetValue (GetValue (GetValue ($ foo) 'a) 'b) 'c)
```

(←@ *accessPath newValue*) [Macro]

Purpose/Behavior: Similar to the @ macro, but used to assign a new value instead of reading a value. Evaluates *newValue*.

Arguments: *accessPath* See Behavior in the @ macro.

newValue Value to be assigned to variable indicated by *accessPath.*

Returns: *newValue*

Example: The following examples show the (←@ accessPath) form, the Parse@ test, and the translation.

1.
```
(←@ foo 1234)
(ParsePut@ (LIST 'foo 1234) 'IV)
(PutValue self 'foo 1234)
```

2.
```
(←@ ($ w) ::left 1234)
(ParsePut@ (LIST ($ w) '::left 1234) 'IV)
(PutClassValue #.($ w) 'left 1234)
```

3.
```
(←@ ($ w) menus DontSave 'Any)
(ParsePut@ (LIST ($ w) 'menus 'DontSave '(QUOTE Any)) 'IV)
(PutValue #.($ w)  'menus  'Any 'DontSave)
```

5.2.2 Support for Changetran

Interlisp uses Changetran to provide an extensive set of facilities for expressing changes to structures, such as push, pushnew, pop, add, change, by using access expressions. You can use any LOOPS access expression in a Changetran context, so that you can now write expressions such as:

```
(push (@ v1) newTop)
(change (@ x)  newValue)
(pushnew (@ colors:,truck) 'red)
(pop (@ ::cv17))
(add (@ x:y:z) 37)
```

The first two are equivalent to:

```
(PushValue self 'v1 (CONS newTop(@ V1)))
(_@ x newValue)
```

This uniform interface allows simpler expressions for changes, and arbitrary extensions through Changetran. See the *Interlisp-D Reference Manual* for more information on Changetran.

5.3 ACCESSING DATA IN CLASSES

5.3 Accessing Data in Classes

A number of functions and methods are available for reading and storing data within classes. Some of these change existing data, and others change the structure of the class by adding variables or properties.

When reading or storing data, some of these functions trigger any active values that are associated with that data. See Chapter 8, Active Values, for a discussion of their behavior.

5.3.1 Metaclass and Property Access

Associated with a class are a metaclass and properties. This section describes the following functions to manipulate their values.

Name	Type	Description
GetClass	Function	Obtains a class's metaclass or properties.

PutClass	Function	Changes the metaclass or class properties of a class.
GetClassOnly	Function	Obtains a class's metaclasses or properties without triggering active values.
PutClassOnly	Function	Changes the metaclass or class properties of a class without triggering active values.
GetClassHere	Function	Obtains a property local to the class.

(**GetClass** *classRec propName*) [Function]

Purpose: Obtains a class's metaclass or properties by following metaclass links.

Behavior: Sends the message **GetClassProp** to *classRec* and passes *propName* as an argument.

Varies according to the arguments.

- If *propName* is NIL, this returns the *class*'s metaclass.
- If *propName* is non-NIL, this looks first in *class* for that property. If it cannot find it there, it looks through *class*'s metaclass links.
- If no property is found, the value of the variable **NotSetValue** is returned.

Arguments: *classRec* Pointer to a class.

propName Property name.

Returns: See Behavior.

Example: The following commands show the variety of returned values.

```
31←(GetClass ($ Window))
 #,($C Class)

32←(GetClass ($ Window) 'doc)
" A LOOPS object which represents a window"

33←(GetClass ($ IconWindow) 'doc)
"An icon window that appears as an irregular shaped image
on the screen -- See the ICONW Library utility"
```

(**PutClass** *classRec newValue propName*) [Function]

Purpose: Changes the metaclass or class properties of a class.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this changes the metaclass of *classRec* to *newValue.* If *newValue* is not a class or the name of a class, this causes a break.
- If *propName* is non-NIL and *classRec* already has this property, this triggers an active value on *propName* if it exists and changes the value of *propName* to *newValue.*
- If *propName* is non-NIL and *classRec* does not have this property, the property is added with the value *newValue*.

Marks the class *classRec* as changed.

Arguments: *classRec* Pointer to a class.

newValue See Behavior.

propName Property name.

Returns: Newly created class object.

Example: The following command changes the **doc** property of class **Datum**:

```
66←(DefineClass 'Datum)
#,($C Datum)

67←(PutClass ($ Datum) '(* this is the updated doc for class Datum) 'doc)
(* this is the updated doc for class Datum)
```

(**GetClassOnly** *classRec propName*) [Function]

Purpose: Obtains a class's metaclass or properties by following superclass links, without triggering active values.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this returns the *classRec*'s metaclass.
- If *propName* is non-NIL, this looks first in *classRec* for that property. If it cannot find it there, it looks through *classRec*'s supers links. This returns the value of the property found without triggering active values.
- If no property is found, the value of the variable **NotSetValue** is returned.

Arguments: *classRec* Pointer to a class.

propName Property name.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetClassOnly ($ IconWindow) 'doc)
```

returns

```
"An icon window that appears as an irregular shaped image
on the screen -- See the ICONW Library utility"
```

(**PutClassOnly** *classRec newValue propName*) [Function]

Purpose: Changes the metaclass or class properties without triggering active values.

Behavior: Varies according to the arguments:

- If *propName* is NIL, this changes the metaclass of *classRec* to *newValue.* If *newValue* is not a class or the name of a class this causes a break.
- If *propName* is non-NIL and *classRec* already has this property, this changes the value of *propName* to *newValue*. Any active values are replaced.
- If *propName* is non-NIL and *classRec* does not have this property, the property is added with the value *newValue*.

The class *classRec* is marked as changed.

Arguments: *classRec* Pointer to a class.

newValue A class or the name of a class.

propName NIL or the name of a class property.

Returns: *newValue*

(**GetClassHere** *classRec propName*) [Function]

Purpose: Obtains property local to class.

Behavior: Gets the class property without triggering active values or inheritance. If there is no local property the value of **NotSetValue** is returned.

Arguments: *classRec* Pointer to a class.

propName NIL or the name of a class property.

Returns: *newValue*

Example: The command

```
(GetClassHere ($ ActiveValue) 'doc)
```

returns

```
#,NotSetValue
```

5.3.2 Class Variable Access

A class variable can be thought of as being shared by all instances of that class and by all instances of any of its subclasses. This section describes how to access class variables with the functions shown in the following table.

Name	Type	Description
GetClassValue	Function	Returns the value of a class variable or property.
PutClassValue	Function	Stores a value in a class variable or property.
GetClassValueOnly	Function	Returns the value of a class variable or property, without triggering active values.
PutClassValueOnly	Function	Stores a value in a class variable or property, without triggering active values.
GetCVHere	Function	Returns the value of a class variable in a particular class without looking for inherited values.
PutCVHere	Function	Stores a class variable locally with a value if it is not local.

(**GetClassValue** *self varName prop*) [Function]

Purpose: Returns the value of a class variable or property.

Behavior: Varies according to the arguments.

If *self* is an instance, the lookup begins at the class of the instance, since instances do not hav class variables stored locally. If *self* is a class, the lookup is in that class.

- If *prop* is NIL, **GetClassValue** returns the value of the class variable *varName*. If *varName* is not found, this breaks.
- If *prop* is non-NIL, **GetClassValue** returns the value of the property *prop*, associated with the class variable *varName*. If the value is an active value, it is activated. If *varName* has no property *prop*, this returns the value of the variable **NoValueFound**.

If the class does not have a class variable *varName*, **GetClassValue** searches through the super classes of the class until it finds *varName*. Since this is rare, class variables are stored only in the class in which they are defined, and the runtime search is necessary.

Arguments: *self* An instance or a class.

varName A class variable name.

prop Property name.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
(← ($ Window) New 'window1)
```

then the command

```
(GetClassValue ($ window1) 'LeftButtonItems)
```

returns the same value as the command

```
(GetClassValue ($ Window) 'LeftButtonItems)
```

The command

```
(GetClassValue ($ Window) 'abcde)
```

breaks. The command

```
(GetClassValue ($ Window) 'LeftButtonItems 'wxyz)
```

returns the value of **NoValueFound**.

(**PutClassValue** *self varName newValue propName*) [Function]

Purpose: Stores a value in a class variable or property.

Behavior: Varies according to the arguments.

If *self* is an instance, the lookup begins at the class of the instance, since instances do not have class variables stored locally. If *self* is a class, the lookup is in that class.

- If *prop* is NIL, **PutClassValue** changes the value of the class variable *varName*.
- If *prop* is non-NIL, **PutClassValue** stores *newValue* as the value of the property, *prop*. If an active value is the current value, it is triggered.

If *varName* is not local to the class, the value is put in the first class in the inheritance list in which *varName* is found. If *varName* is not found, a break occurs.

Arguments: *self* An instance or a class.

varName A class variable name.

newValue A new value.

propName Property name.

Returns: *newValue*

Example: Given that

```
(← ($ Window) New 'window1)
```

then the command

```
(PutClassValue ($ window1) 'LeftButtonItems 2 'number)
```

adds the property number with the value 2 to the class variable **LeftButtonItems** of the class **Window.** The following command performs the same action.

```
(PutClassValue ($ Window) 'LeftButtonItems 2 'number)
```

(**GetClassValueOnly** *classRec varName prop*) [Function]

Purpose: Returns the value of a class variable or property, without triggering active values.

Behavior: Similar to **GetClassValue**, with the following exceptions:

- If **GetClassValueOnly** finds that the value is an active value, the active value is returned without being triggered.
- If *prop* is non-NIL and is not found, **GetClassValueOnly** returns the value of the variable **NotSetValue**.

Arguments: See **GetClassValue**.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetClassValueOnly ($ Window) 'abcde)
```

breaks. The command

```
(GetClassValueOnly ($ Window) 'LeftButtonItems 'wxyz)
```

returns the value of **NotSetValue.**

(**PutClassValueOnly** *self varName newValue propName*) [Function]

Purpose: Stores the value of a class variable or property, without triggering active values.

Behavior: Similar to **PutClassValue**, except that **PutClassValueOnly** does not trigger an active value, but replaces it with *newValue*.

Arguments: See **PutClassValue** .

Returns: Used for side effect only.

(**GetCVHere** *classRec varName propName*) [Function]

Purpose: Returns the value of a class variable in a particular class without looking for inherited values.

Behavior: Returns the value of the class variable *varName*, or the *propName* property if *propName* is non-NIL.

If the value is an active value, it is returned without being triggered.

If there is no *varName* (or *propName*), this returns the value of the variable **NotSetValue**.

Arguments:
classRec Must be a class.
varName A class variable name.
propName Property name.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetCVHere ($ NonRectangularWindow) 'LeftButtonItems)
```

returns

```
#,NotSetValue
```

The command

```
(GetCVHere ($ Window) 'LeftButtonItems)
```

returns

```
((Update (QUOTE Update)...)
```

(**PutCVHere** *self varName value*) [Function]

Purpose: Puts a class variable locally with a value if it is not local.

Behavior: Calls (**AddCV** *self varName value*).

Arguments:
self An instance or a class.
varName A class variable name.
value Value for the class variable.

Returns: *value*

5.3.3 Instance Variable Access

An instance variable can be thought of as being local to each instance of a class. The class defines what instance variables and their default values will be in an instance. This section describes the functions that manipulate the default values in the class.

See the *LOOPS Library Modules Manual* for interaction with Masterscope.

Name	Type	Description
GetClassIV	Function	Gets the default value of an instance variable or associated property as defined in a class or one of its supers.
GetClassIVHere	Function	Gets the default value of an instance variable or associated property as defined in a class.

PutClassIV	Function	Changes the default value for an instance variable in a class.

(**GetClassIV** *self varName prop*) [Function]

Purpose: Gets the default value of an instance variable or associated property as defined in a class or one of its supers.

Behavior: If *self* is not bound to a class, an error occurs.

Searches through the supers of *self* to find *varName* or *prop*.

- If *prop* is NIL, this returns the default value for *varName*.
- If *prop* is non-NIL, this returns its default value.

If the default value is an active value, it is returned without being triggered.

Arguments:

self	Must be bound to a class.
varName	The name of an instance variable.
prop	Name of a property associated with *varName*.

Returns: Value depends on the arguments; see Behavior.

Example: The commands

```
(GetClassIV ($ Window) 'window)
(GetClassIV ($ NonRectangularWindow) 'window)
```

both return

```
#,($AV LispWindowAV ...)
```

(**GetClassIVHere** *self varName prop*) [Function]

Purpose: Gets the default value of an instance variable or associated property as defined in a class.

Behavior: Similar to **GetClassIV**. This does not search the super classes of *self* for *varName*. If *varName* or *prop* is not local to *self*, this returns the value of **NotSetValue**.

Arguments:

self	Pointer to a class.
varName	Name of an instance variable.
prop	Name of a property associated with *varName*.

Returns: The default value of *varName* or *prop* or **NotSetValue**.

Example: The command

```
(GetClassIVHere ($ Window) 'window)
```

returns

```
#,($AV LispWindowAV ...)
```

the command

```
(GetClassIVHere ($ NonRectangularWindow) 'window)
```

returns

```
#,NotSetValue
```

(**PutClassIV** *self varName newValue propName*) [Function]

Purpose: Changes the default value for an IV in a class.

Behavior: If *self* is not a class that contains the instance variable *varName*, an error occurs.

- If *propName* is NIL, the default value for the instance variable *varName* is changed to *newValue*.
- If *propName* is non-NIL, the default value for it is changed to *newValue*.

Arguments:

self	Must be a class that contains the instance variable *varName*.
varName	An instance variable name.
newValue	The new default value.
propName	Property name.

Returns: *newValue* (used for side effect only).

Example: After the commands

```
68←(DefineClass 'Datum)
#,($C Datum)

69←(← ($ Datum) AddIV 'id# NIL)
id#
```

the following command changes the default value of the instance variable **id#** to '(7) for all new instances of the class **Datum**:

```
70←(PutClassIV ($ Datum) 'id# '(7))
(7)
```

[This page intentionally left blank]