TABLE OF CONTENTS

2. LITATOMS

A "litatom" (for "literal atom") is an object which conceptually consists of a print name, a value, a function definition, and a property list. In some Lisp dialects, litatoms are also known as "symbols."

A litatom is read as any string of non-delimiting characters that cannot be interpreted as a number. The syntatic characters that delimit litatoms are called separator or break characters (see page 25.33) and normally are space, end-of-line, line-feed, ((left paren),) (right paren), " (double quote), [(left bracket), and] (right bracket). However, any character may be included in a litatom by preceding it with the character %. Here are some examples of litatoms:

A wxyz 23SKIDDOO %] 3.1415+17

Long% Litatom% With% Embedded% Spaces

(LITATOM *X*) [Function]

Returns **T** if *X* is a litatom, **NIL** otherwise. Note that a number is not a litatom.

(LITATOM NIL) = T.

(ATOM *X*) [Function]

Returns **T** if *X* is an atom (i.e. a litatom or a number); **NIL** otherwise.

Warning: **(ATOM *X*)** is **NIL** if *X* is an array, string, etc. In many dialects of Lisp, the function **ATOM** is defined equivalent to the Interlisp function **NLISTP**.

(ATOM NIL) = T.

Litatoms are printed by **PRINT** and **PRIN2** as a sequence of characters with %'s inserted before all delimiting characters (so that the litatom will read back in properly). Litatoms are printed by **PRIN1** as a sequence of characters without these extra %'s. For example, the litatom consisting of the five characters **A**, **B**, **C**, **(**, and **D** will be printed as **ABC%(D** by **PRINT** and **ABC(D** by **PRIN1**.

Litatoms can also be constructed by **PACK**, **PACK***, **SUBATOM**, **MKATOM**, and **GENSYM** (which uses **MKATOM**).

Litatoms are unique. In other words, if two litatoms print the same, they will *always* be **EQ**. Note that this is *not* true for strings, large integers, floating point numbers, and lists; they all can print the same without being **EQ**. Thus if **PACK** or **MKATOM** is given a list of characters corresponding to a litatom that already exists, they return a pointer to that litatom, and do *not* make a new litatom. Similarly, if the read program is given as input a sequence of characters for which a litatom already exists, it returns a pointer to that litatom. Note: Interlisp is different from other Lisp dialects which allow "uninterned" litatoms.

Note: Litatoms are limited to 255 characters in Interlisp-D; 127 characters in Interlisp-10. Attempting to create a larger litatom either via **PACK** or by typing one in (or reading from a file) will cause an error, **ATOM TOO LONG**.

2.1 Using Litatoms as Variables

Litatoms are commonly used as variables. Each litatom has a "top level" variable binding, which can be an arbitrary Interlisp object. Litatoms may also be given special variable bindings within **PROG**s or function calls, which only exist for the duration of the function. When a litatom is evaluated, the "current" variable binding is returned. This is the most recent special variable binding, or the top level binding if the litatom has not been rebound. **SETQ** is used to change the current binding. For more information on variable bindings in Interlisp, see page 11.1.

Note: The compiler (page 18.1) treats variables somewhat differently than the interpreter, and the user has to be aware of these differences when writing functions that will be compiled. For example, variable references in compiled code are not checked for **NOBIND**, so compiled code will not generate unbound atom errors. In general, it is better to debug interpreted code, before compiling it for speed. The compiler offers some facilities to increase the efficiency of variable use in compiled functions. Global variables (page 18.4) can be defined so that the entire stack is not searched at each variable reference. Local variables (page 18.5) allow compiled functions to access variable bindings which are not on the stack, which reduces variable conflicts, and also makes variable lookup faster.

By convention, a litatom whose top level binding is to the litatom **NOBIND** is considered to have no top level binding. If a litatom has no local variable bindings, and its top level value is **NOBIND**, attempting to evaluate it will cause an unbound atom error.

The two litatoms **T** and **NIL** always evaluate to themselves. Attempting to change the binding of **T** or **NIL** with the functions below will generate the error **ATTEMPT TO SET T** or **ATTEMPT TO SET NIL**.

The following functions (except **BOUNDP**) will also generate the error **ARG NOT LITATOM**, if not given a litatom.

(BOUNDP *VAR***)** [Function]

Returns **T** if *VAR* has a special variable binding (even if bound to **NOBIND**), or if *VAR* has a top level value other than **NOBIND**; otherwise **NIL**. In other words, if *X* is a litatom, **(EVAL** *X***)** will cause an **UNBOUND ATOM** error if and only if **(BOUNDP** *X***)** returns **NIL**.

(SET *VAR VALUE***)** [Function]

Sets the "current" variable binding of *VAR* to *VALUE*, and returns *VALUE*.

Note that **SET** is a normal lambda spread function, so both *VAR* and *VALUE* are evaluated before it is called. Thus, if the value of **X** is **B**, and the value of **Y** is **C**, then **(SET X Y)** would result in **B** being set to **C**, and **C** being returned as the value of **SET**.

(SETQ *VAR VALUE***)** [NLambda NoSpread Function]

Nlambda version of **SET**; *VAR* is not evaluated, *VALUE* is. Thus if the value of **X** is **B** and the value of **Y** is **C**, **(SETQ X Y)** would result in **X** (not **B**) being set to **C**, and **C** being returned.

Note: Since **SETQ** is an nlambda, *neither* argument is evaluated during the calling process. However, **SETQ** itself calls **EVAL** on its second argument. As a result, typing **(SETQ VAR FORM)** and **SETQ(VAR FORM)** to the Interlisp executive is equivalent: in both cases **VAR** is not evaluated, and **FORM** is.

(SETQQ *VAR VALUE***)** [NLambda Function]

Like **SETQ** except that neither argument is evaluated, e.g., **(SETQQ X (A B C))** sets **X** to **(A B C)**.

(PSETQ VAR_1 $VALUE_1$ ... VAR_N $VALUE_N$**)** [Macro]

Does a multiple **SETQ** of VAR_1 (unevaluated) to the value of $VALUE_1$, VAR_2 to the value of $VALUE_2$, etc. All of the $VALUE_i$ terms are evaluated before any of the assignments. Therefore, **(PSETQ A B B A)** can be used to swap the values of the variables **A** and **B**.

(GETTOPVAL *VAR***)** [Function]

Returns the top level value of *VAR* (even if **NOBIND**), regardless of any intervening local bindings.

(SETTOPVAL *VAR VALUE***)** [Function]

Sets the top level value of *VAR* to *VALUE*, regardless of any intervening bindings, and returns *VALUE*.

A major difference between various Interlisp implementations is the way that variable bindings are implemented. Interlisp-10 and Interlisp-Jerico use what is called "shallow" binding. Interlisp-D and Interlisp-VAX use what is called "deep" binding.

In a deep binding system, a variable is bound by saving on the stack the variable's new value. When a variable is accessed, its value is found by searching the stack for the most recent binding. If the variable is not found on the stack, the top level binding is retrieved from a "value cell" associated with the variable.

In a "shallow" binding system, a variable is bound by saving on the stack the variable name and the variable's old value and putting the new value in the variable's value cell. When a variable is accessed, its value is always found in its value cell.

GETTOPVAL and **SETTOPVAL** are less efficient in a shallow binding system, because they have to search the stack for rebindings; it is more economical to simply rebind variables. In a deep binding system, **GETTOPVAL** and **SETTOPVAL** are very efficient since they do not have to search the stack, but can simply access the value cell directly.

GETATOMVAL and **SETATOMVAL** can be used to access a variable's value cell, in either a shallow or deep binding system.

(GETATOMVAL *VAR***)** [Function]

Returns the value in the value cell of *VAR*. In a shallow binding system, this is the same as **(EVAL** *ATM***)**, or simply *VAR*. In a deep binding system, this is the same as **(GETTOPVAL** *VAR***)**.

(SETATOMVAL *VAR VALUE***)** [Function]

Sets the value cell of *VAR* to *VALUE*. In a shallow binding system, this is the same as **SET**; in a deep binding system, this is the same as **SETTOPVAL**.

2.2 Function Definition Cells

Each litatom has a function definition cell, which is accessed when a litatom is used as a function. The mechanism for accessing and setting the function definition cell of a litatom is described on page 10.9.

2.3 Property Lists

Each litatom has an associated property list, which allows a set of named objects to be associated with the litatom. A property list associates a name, known as a "property name" or "property", with an abitrary object, the "property value" or simply "value". Sometimes the phrase "to store on the property *X*" is used, meaning to place the indicated information on a property list under the property name *X*.

Property names are usually litatoms or numbers, although no checks are made. However, the standard property list functions all use **EQ** to search for property names, so they may not work with non-atomic property names. Note that the same object can be used as both a property name and a property value.

Note: Many litatoms in the system already have property lists, with properties used by the compiler, the break package, DWIM, etc. Be careful not to clobber such system properties. The variable **SYSPROPS** is a list of property names used by the system.

The functions below are used to manipulate the propert lists of litatoms. Except when indicated, they generate the error **ARG NOT LITATOM**, if given an object that is not a litatom.

(GETPROP *ATM PROP***)** [Function]

Returns the property value for *PROP* from the property list of *ATM*. Returns **NIL** if *ATM* is not a litatom, or *PROP* is not found. Note that **GETPROP** also returns **NIL** if there is an occurrence of *PROP* but the corresponding property value is **NIL**; this can be a source of program errors.

Note: **GETPROP** used to be called **GETP**.

(PUTPROP *ATM PROP VAL***)** [Function]

Puts the property *PROP* with value *VAL* on the property list of *ATM*. *VAL* replaces any previous value for the property *PROP* on this property list. Returns *VAL*.

(ADDPROP *ATM PROP NEW FLG***)** [Function]

Adds the value *NEW* to the list which is the value of property *PROP* on the property list of *ATM*. If *FLG* is **T**, *NEW* is **CONS**ed onto the front of the property value of *PROP*, otherwise it is **NCONC**ed on the end (using **NCONC1**). If *ATM* does not have a property *PROP*, or the value is not a list, then the effect is the same as **(PUTPROP** *ATM PROP* **(LIST** *NEW***))**. **ADDPROP** returns the (new) property value. Example:

```
←(PUTPROP 'POCKET 'CONTENTS NIL)
NIL
←(ADDPROP 'POCKET 'CONTENTS 'COMB)
(COMB)
←(ADDPROP 'POCKET 'CONTENTS 'WALLET)
(COMB WALLET)
```

(REMPROP *ATM PROP***)** [Function]

Removes all occurrences of the property *PROP* (and its value) from the property list of *ATM*. Returns *PROP* if any were found, otherwise **NIL**.

(REMPROPLIST *ATM PROPS***)** [Function]

Removes all occurrences of all properties on the list *PROPS* (and their corresponding property values) from the property list of *ATM*. Returns **NIL**.

(CHANGEPROP *X PROP1 PROP2***)** [Function]

Changes the property name of property *PROP1* to *PROP2* on the property list of *X*, (but does not affect the value of the property). Returns *X*, unless *PROP1* is not found, in which case it returns **NIL**.

(PROPNAMES *ATM***)** [Function]

Returns a list of the property names on the property list of *ATM*.

(DEFLIST *L PROP***)** [Function]

Used to put values under the same property name on the property lists of several litatoms. *L* is a list of two-element lists. The first element of each is a litatom, and the second element is the property value for the property *PROP*. Returns **NIL**. For example,

```
(DEFLIST '( (FOO MA) (BAR CA) (BAZ RI) ) 'STATE)
```

puts **MA** on **FOO**'s **STATE** property, **CA** on **BAR**'s **STATE** property, and **RI** on **BAZ**'s **STATE** property.

Property lists are conventionally implemented as lists of the form

$(NAME_1\ VALUE_1\ NAME_2\ VALUE_2 \ldots)$

although the user can store anything as the property list of a litatom. However, the functions which manipulate property lists observe this convention by searching down the property lists two **CDR**s at a time. Most of these functions also generate an error, **ARG NOT LITATOM**, if given an argument which is not a litatom, so they cannot be used directly on lists. (**LISTPUT**, **LISTPUT1**, **LISTGET**, and **LISTGET1** are functions similar to **PUTPROP** and **GETPROP** that work directly on lists. See page 3.16.) The property lists of litatoms can be directly accessed with the following functions:

(GETPROPLIST *ATM*) [Function]

Returns the property list of *ATM*.

(SETPROPLIST *ATM LST*) [Function]

If *ATM* is a litatom, sets the property list of *ATM* to be *LST*, and returns *LST* as its value.

(GETLIS *X PROPS*) [Function]

Searches the property list of *X*, and returns the property list as of the first property on *PROPS* that it finds. For example,

← (GETPROPLIST 'X)
(PROP1 A PROP3 B A C)
← (GETLIS 'X '(PROP2 PROP3))
(PROP3 B A C)

Returns **NIL** if no element on *PROPS* is found. *X* can also be a list itself, in which case it is searched as described above. If *X* is not a litatom or a list, returns **NIL**.

2.4 Print Names

Each litatom has a print name, a string of characters that uniquely identifies that litatom. The term "print name" has been extended, however, to refer to the characters that are output when any object is printed. In Interlisp, all objects have print names, although only litatoms and strings have their print name explicitly stored. This section describes a set of functions which can be used to access and manipulate the print names of any object, though they are primarily used with the print names of litatoms.

The print name of an object is those characters that are output when the object is printed using **PRIN1**, e.g., the print name of the litatom **ABC%(D** consists of the five characters **ABC(D**. The

print name of the list **(A B C)** consists of the seven characters **(A B C)** (two of the characters are spaces).

Sometimes we will have occasion to refer to a "**PRIN2**-name." The **PRIN2**-name of an object is those characters output when the object is printed using **PRIN2**. Thus the **PRIN2**-name of the litatom **ABC%(D** is the *six* characters **ABC%(D**. Note that the **PRIN2**-name depends on what readtable is being used (see page 25.33), since this determines where %'s will be inserted. Many of the functions below allow either print names or **PRIN2**-names to be used, as specified by *FLG* and *RDTBL* arguments. If *FLG* is **NIL**, print names are used. Otherwise, **PRIN2**-names are used, computed with respect to the readtable *RDTBL* (or the current readtable, if *RDTBL* = **NIL**).

Note: The print name of an integer depends on the setting of **RADIX** (page 25.13). The functions described in this section (**UNPACK**, **NCHARS**, etc.) define the print name of an integer as though the radix was 10, so that **(PACK (UNPACK 'X9))** will always be **X9** (and not **X11**, if **RADIX** is set to 8). However, integers will still be *printed* by **PRIN1** using the current radix. The user can force these functions to use print names in the current radix by changing the setting of the variable **PRXFLG** (page 25.14).

(MKATOM *X***)** [Function]

Creates and returns an atom whose print name is the same as that of the string *X* or, if *X* isn't a string, the same as that of **(MKSTRING** *X***)**. Examples:

(MKATOM '(A B C)) => %(A% B% C%)

(MKATOM "1.5") => 1.5

Note that the last example returns a number, not a litatom. It is a deeply-ingrained feature of Interlisp that no litatom can have the print name of a number.

(SUBATOM *X N M***)** [Function]

Equivalent to **(MKATOM (SUBSTRING** *X N M***))**, but does not make a string pointer (see page 4.3). Returns an atom made from the *N*th through *M*th characters of the print name of *X*. If *N* or *M* are negative, they specify positions counting backwards from the end of the print name. Examples:

(SUBATOM "FOO1.5BAR" 4 6) => 1.5

(SUBATOM '(A B C) 2 -2) => A% B% C

(PACK *X***)** [Function]

If *X* is a list of atoms, **PACK** returns a single atom whose print name is the concatenation of the print names of the atoms in *X*.

If the concatenated print name is the same as that of a number, **PACK** will return that number. For example,

(PACK '(A BC DEF G)) = > ABCDEFG

(PACK '(1 3.4)) = > 13.4

(PACK '(1 E -2)) = > .01

Although *X* is usually a list of atoms, it can be a list of arbitrary Interlisp objects. The value of **PACK** is still a single atom whose print name is the concatenation of the print names of all the elements of *X*, e.g.,

(PACK '((A B) "CD")) = > %(A% B%)CD

If *X* is not a list or **NIL**, **PACK** generates an error, **ILLEGAL ARG**.

(PACK* X_1 X_2 ... X_N) [NoSpread Function]

Nospread version of **PACK** that takes an arbitrary number of arguments, instead of a list. Examples:,

(PACK* 'A 'BC 'DEF 'G) = > ABCDEFG

(PACK* 1 3.4) = > 13.4

(UNPACK *X FLG RDTBL***)** [Function]

Returns the print name of *X* as a list of single-characters atoms, e.g.,

(UNPACK 'ABC5D) = > (A B C 5 D)

(UNPACK "ABC(D") = > (A B C %(D)

If *FLG* = **T**, the **PRIN2**-name of *X* is used (computed with respect to *RDTBL*), e.g.,

(UNPACK "ABC(D" T) = > (%" A B C %(D %")

(UNPACK 'ABC%(D" T) = > (A B C %% %(D)

Note: **(UNPACK** *X***)** performs *N* **CONS**es, where *N* is the number of characters in the print name of *X*.

(DUNPACK *X SCRATCHLIST FLG RDTBL***)** [Function]

A destructive version of **UNPACK** that does not perform any **CONS**es but instead reuses the list *SCRATCHLIST*. If the print name is too long to fit in *SCRATCHLIST*, **DUNPACK** will extend it. If *SCRATCHLIST* is not a list, **DUNPACK** returns **(UNPACK** *X FLG RDTBL***)**.

(NCHARS *X FLG RDTBL***)** [Function]

Returns the number of characters in the print name of *X*. If *FLG* = **T**, the **PRIN2**-name is used. For example,

(NCHARS 'ABC) = > 3

(NCHARS "ABC" T) = > 5

Note: **NCHARS** works most efficiently on litatoms and strings, but can be given any object.

(NTHCHAR *X N FLG RDTBL*) [Function]

Returns the *N*th character of the print name of *X* as an atom. *N* can be negative, in which case it counts from the end of the print name, e.g., -1 refers to the last character, -2 next to last, etc. If *N* is greater than the number of characters in the print name, or less than minus that number, or 0, **NTHCHAR** returns **NIL**. Examples:

(NTHCHAR 'ABC 2) = > B

(NTHCHAR 15.6 2) = > 5

(NTHCHAR 'ABC%(D -3 T) = > %%

(NTHCHAR "ABC" 2) = > B

(NTHCHAR "ABC" 2 T) = > A

Note: **NTHCHAR** and **NCHARS** work much faster on objects that actually have an internal representation of their print name, i.e., litatoms and strings, than they do on numbers and lists, as they do not have to simulate printing.

(L-CASE *X FLG*) [Function]

Returns a lower case version of *X*. If *FLG* is **T**, the first letter is capitalized. If *X* is a string, the value of **L-CASE** is also a string. If *X* is a list, **L-CASE** returns a new list in which **L-CASE** is computed for each corresponding element and non-**NIL** tail of the original list. Examples:

(L-CASE 'FOO) = > foo

(L-CASE 'FOO T) = > Foo

(L-CASE "FILE NOT FOUND" T) = > "File not found"

(L-CASE '(JANUARY FEBRUARY (MARCH "APRIL")) T)
= > '(January February (March "April"))

(U-CASE *X*) [Function]

Similar to **L-CASE**, except returns the upper case version of *X*.

(U-CASEP *X*) [Function]

Returns **T** if *X* contains no lower case letters; **NIL** otherwise.

(GENSYM *PREFIX* — — — —) [Function]

Returns a litatom of the form **Xnnnn**, where **X** = *PREFIX* (or **A** if *PREFIX* is **NIL**) and **nnnn** is an integer. Thus, the first one

generated is **A0001**, the second **A0002**, etc. The integer suffix is always at least four characters long, but it can grow beyond that. For example, the next litatom produced after **A9999** would be **A10000**. **GENSYM** provides a way of generating litatoms for various uses within the system.

GENNUM [Variable]

The value of **GENNUM**, initially **0**, determines the next **GENSYM**, e.g., if **GENNUM** is set to **23**, **(GENSYM) = A0024**.

The term "gensym" is used to indicate a litatom that was produced by the function **GENSYM**. Litatoms generated by **GENSYM** are the same as any other litatoms: they have property lists, and can be given function definitions. Note that the litatoms are not guaranteed to be new. For example, if the user has previously created **A0012**, either by typing it in, or via **PACK** or **GENSYM** itself, then if **GENNUM** is set to **11**, the next litatom returned by **GENSYM** will be the **A0012** already in existence.

(MAPATOMS *FN***)** [Function]

Applies *FN* (a function or lambda expression) to every litatom in the system. Returns **NIL**

For example,

(MAPATOMS (FUNCTION (LAMBDA(X)
(if (GETD X) then (PRINT X)]

will print every litatom with a function definition.

Note: In some implementations of Interlisp, unused litatoms may be garbage collected, which can effect the action of **MAPATOMS**.

(APROPOS *STRING ALLFLG QUIETFLG OUTPUT***)** [Function]

APROPOS scans all litatoms in the system for those which have *STRING* as a substring and prints them on the terminal along with a line for each relevant item defined for each selected atom. Relevant items are (1) function definitions, for which only the arglist is printed, (2) dynamic variable values, and (3) non-null property lists. **PRINTLEVEL** (page 25.11) is set to **(3 . 5)** when **APROPOS** is printing.

If *ALLFLG* is **NIL**, then atoms with no relevant items and "internal" atoms are omitted ("internal" currently means those litatoms whose print name begins with a \ or those litatoms produced by **GENSYM**). If *ALLFLG* is a function (i.e., **(FNTYP** *ALLFLG***)** is non-**NIL**), then it is used as a predicate on atoms selected by the substring match, with value **NIL** meaning to omit the atom. If *ALLFLG* is any other non-**NIL** value, then no atoms are omitted.

If *QUIETFLG* is non-**NIL**, then no printing at all is done, but instead a list of the selected atoms is returned.

If *OUTPUT* is non-**NIL**, the printing will be directed to *OUTPUT* (which should be a stream open for output) instead of to the terminal stream.

2.5 Characters and Character Codes

Characters may be represented in two ways: as single-character atoms, or as integer character codes. In many situations, it is more efficient to use character codes, so Interlisp provides parallel functions for both representations.

Interlisp-D uses the 16-bit NS character set, described in the document *Character Code Standard* [Xerox System Integration Standards, XSIS 058404, April 1984]. Legal character codes range from 0 to 65535. The NS (Network Systems) character encoding encompasses a much wider set of available characters than the 8-bit character standards (such as ASCII), including characters comprising many foreign alphabets and special symbols. For instance, Interlisp-D supports the display and printing of the following:

Le système d'information Xerox 11xx est remarquablement polyglotte.

Das Xerox 11xx Kommunikationssystem bietet merkwürdige multilinguale Nutzmöglichkeiten.

$M \models \Box[w] \Leftrightarrow \forall v \text{ with } Rwv: M \models [v]$

These characters can be used in strings, litatom print names, symbolic files, or anywhere else 8-bit characters could be used. All of the standard string and print name functions (**RPLSTRING**, **GNC**, **NCHARS**, **STRPOS**, etc.) accept litatoms and strings containing NS characters. For example:

←(STRPOS "char" "this is an 8-bit character string")
18
←(STRPOS "char" "celui-ci comporte des charactères NS")
23

In almost all cases, a program does not have to distinguish between NS characters or 8-bit characters. The exception to this rule is the handling of input/output operations (see page 25.22).

The function **CHARCODE** (page 2.13) provides a simple way to create individual NS characters codes. The VirtualKeyboards library package provides a set of virtual keyboards that allow keyboard or mouse entry of NS characters.

(PACKC *X*) [Function]

Similar to **PACK** except *X* is a list of character codes. For example,

(PACKC '(70 79 79)) = > FOO

(CHCON *X FLG RDTBL*) [Function]

Like **UNPACK**, except returns the print name of *X* as a list of character codes. If *FLG* = **T**, the **PRIN2**-name is used. For example,

(CHCON 'FOO) = > (70 79 79)

(DCHCON *X SCRATCHLIST FLG RDTBL*) [Function]

Similar to **DUNPACK**.

(NTHCHARCODE *X N FLG RDTBL*) [Function]

Similar to **NTHCHAR**, except returns the character code of the *N*th character of the print name of *X*. If *N* is negative, it is interpreted as a count backwards from the end of *X*. If the absolute value of *N* is greater than the number of characters in *X*, or 0, then the value of **NTHCHARCODE** is **NIL**.

If *FLG* is **T**, then the **PRIN2**-name of *X* is used, computed with respect to the readtable *RDTBL*

(CHCON1 *X*) [Function]

Returns the character code of the first character of the print name of *X*; equal to **(NTHCHARCODE *X* 1)**.

(CHARACTER *N*) [Function]

N is a character code. Returns the atom having the corresponding single character as its print name.

(CHARACTER 70) = > F

(FCHARACTER *N*) [Function]

Fast version of **CHARACTER** that compiles open.

The following function makes it possible to gain the efficiency that comes from dealing with character codes without losing the symbolic advantages of character atoms:

(CHARCODE *CHAR*) [NLambda Function]

Returns the character code specified by *CHAR* (unevaluated). If *CHAR* is a one-character atom or string, the corresponding character code is simply returned. Thus, **(CHARCODE A)** is 65, **(CHARCODE 0)** is 48. If *CHAR* is a multi-character litatom or string, it specifies a character code as described below. If *CHAR* is **NIL**, **CHARCODE** simply returns **NIL**. Finally, if *CHAR* is a list

structure, the value is a copy of *CHAR* with all the leaves replaced by the corresponding character codes. For instance, **(CHARCODE (A (B C))) = > (65 (66 67))**.

If a character is specified by a multi-character litatom or string, **CHARCODE** interprets it as follows:

CR, SPACE, etc. The variable **CHARACTERNAMES** contains an association list mapping special litatoms to character codes. Among the characters defined this way are **CR** (13), **LF** (10), **SPACE** or **SP** (32), **ESCAPE** or **ESC** (27), **BELL** (7), **BS** (8), **TAB** (9), **NULL** (0), and **DEL** (127). The litatom **EOL** maps into the appropriate End-Of-Line character code in the different Interlisp implementations (31 in Interlisp-10, 13 in Interlisp-D, 10 in Interlisp-VAX). Examples:

(CHARCODE SPACE) = > 32

(CHARCODE CR) = > 13

CHARSET,CHARNUM
CHARSET-CHARNUM If the character specification is a litatom or string of the form *CHARSET,CHARNUM* or *CHARSET-CHARNUM*, the character code for the character number *CHARNUM* in the character set *CHARSET* is returned.

The 16-bit NS character encoding is divided into a large number of "character sets." Each 16-bit character can be decoded into a character set (an integer from 0 to 254 inclusive) and a character number (also an integer from 0 to 254 inclusive). *CHARSET* is either an *octal* number, or a litatom in the association list **CHARACTERSETNAMES** (which defines the character sets for **GREEK, CYRILLIC**, etc.).

CHARNUM is either an *octal* number, a single-character litatom, or a litatom from the association list **CHARACTERNAMES**. Note that if *CHARNUM* is a single-digit number, it is interpreted as the character "2", rather than as the octal number 2. Examples:

(CHARCODE 12,6) = > 2566

(CHARCODE 12,SPACE) = > 2592

(CHARCODE GREEK,A) = > 9793

↑ *CHARSPEC* (control chars) If the character specification is a litatom or string of one of the forms above, preceeded by the character "↑", this indicates a "control character," derived from the normal character code by clearing the seventh bit of the character code (normally set). Examples:

(CHARCODE ↑ A) = > 1

(CHARCODE ↑ GREEK,A) = > 9729

#CHARSPEC (meta chars) If the character specification is a litatom or string of one of the forms above, preceeded by the character "#", this indicates a "meta character," derived from the normal character code by

setting the eighth bit of the character code (normally cleared). ↑ and # can both be set at once. Examples:

(CHARCODE #A) => 193

(CHARCODE # ↑ GREEK,A) => 9857

A **CHARCODE** form can be used wherever a structure of character codes would be appropriate. For example:

```
(FMEMB (NTHCHARCODE X 1) (CHARCODE (CR LF SPACE ↑ A)))
(EQ (READCCODE FOO) (CHARCODE GREEK,A))
```

There is a macro for **CHARCODE** which causes the character-code structure to be constructed at compile-time. Thus, the compiled code for these examples is exactly as efficient as the less readable:

```
(FMEMB (NTHCHARCODE X 1) (QUOTE (13 10 32 1)))
(EQ (READCCODE FOO) 9793)
```

(SELCHARQ *E CLAUSE*$_1$ ... *CLAUSE*$_N$ *DEFAULT*) [Macro]

Similar to **SELECTQ** (page 9.6), except that the selection keys are determined by applying **CHARCODE** (instead of **QUOTE**) to the key-expressions. If the value of *E* is a character code or **NIL** and it is **EQ** or **MEMB** to the result of applying **CHARCODE** to the first element of a clause, the remaining forms of that clause are evaluated. Otherwise, the default is evaluated.

Thus

```
(SELCHARQ (BIN FOO)
  ((SPACE TAB) (FUM))
  (( ↑ D NIL) (BAR))
  (a (BAZ))
  (ZIP))
```

is exactly equivalent to

```
(SELECTQ (BIN FOO)
  ((32 9) (FUM))
  ((4 NIL) (BAR))
  (97 (BAZ))
  (ZIP))
```

Furthermore, **SELCHARQ** has a macro definition such that it always compiles as an equivalent **SELECTQ**.