3. COMMON LISP/INTERLISP-D INTEGRATION

NOTE: Chapter 3 is organized to correspond to the original *Interlisp-D Reference Manual*, and explains changes related to how Common Lisp affects Interlisp-D in your Lisp software development environment. To make it easy to use this chapter with the *IRM,* information is organized by *IRM* volume and section numbers. Section headings from the *IRM* are maintained to aid in cross-referencing.

Lyric information as well as Medley release enhancements are included. Medley additions are indicated with revision bars in the right margin.

VOLUME I—LANGUAGE

Chapter 2 Litatoms

(2.1)

What Interlisp calls a "LITATOM" is the same as what Common Lisp calls a "SYMBOL." Symbols are partitioned into separate name spaces called packages. When you type a string of characters, the resulting symbol is searched for in the "current package." A colon in the symbol separates a package name from a symbol name; for example, the string of characters "CL:AREF" denotes the symbol AREF accessible in the package CL. For a full discussion, see Guy Steele's *Common Lisp, the Language*.

All the functions in this section that create symbols do so in the INTERLISP package (IL), which is also where all the symbols in the *Interlisp-D Reference Manual* are found. Note that this is true even in cases where you might not expect it. For example, U-CASE returns a symbol in the INTERLISP package, even when its argument is in some other package; similarly with L-CASE and SUBATOM. In most cases, this is the right thing for an Interlisp program; e.g., U-CASE in some sense returns a "canonical" symbol that one might pass to a SELECTQ, regardless of which executive it was typed in. However, to perform symbol manipulations that preserve package information, you should use the appropriate Common Lisp functions (See *Common Lisp the Language*, Chapter 11, Packages and Chapter 18, Strings).

Symbols read under an old Interlisp readtable are also searched for in the INTERLISP package. See Section 25.8, Readtables, for more details.

Section 2.1 Using Litatoms as Variables

(I:2.3)

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

The Interlisp interpreter has been modified to consider any symbol bound to the distinguished symbol **NOBIND** to be unbound. It will signal an UNBOUND-VARIABLE condition on encountering references to such symbols. In prior releases, the interpreter only considered a symbol unbound if it had no dynamic binding and in addition its top-level value was **NOBIND**.

For most user code, this change has no effect, as it is unusual to bind a variable to the particular value **NOBIND** and still deliberately want the variable to be considered bound. However, it is a particular problem when an interpreted Interlisp function is passed to the function **MAPATOMS**. Since **NOBIND** is a symbol, it will eventually be passed as an argument to the interpreted function. The first reference to that argument within the function will signal an error.

A work-around for this problem is to use a Common Lisp function instead. Calls to this function will invoke the Common Lisp interpreter which will treat the argument as a local, not special, variable. Thus, no error will be signaled. Alternatively, one could include the argument to the Interlisp function in a **LOCALVARS** declaration and then compile the function before passing it to **MAPATOMS**. This has the advantage of significantly speeding up the **MAPATOMS** call.

Section 2.3 Property Lists

(I:2.6)

The value returned from the function **REMPROP** has been changed in one case:

(**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 (**T** if *PROP* is **NIL**), otherwise **NIL**.

Section 2.4 Print Names

(I:2.7)

The print functions now qualify the name of a symbol with a package prefix if the symbol is not accessible in the current package. The Interlisp "PRIN1" print name of a symbol does not include the package name.

(I:2.10)

The **GENSYM** function in Interlisp creates symbols interned in the INTERLISP package. The Common Lisp **CL:GENSYM** function creates uninterned symbols.

(I:2.11)

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

See the note for **BOUNDP** above.

Section 2.5 Characters

A "character" in Interlisp is different from the type "character" in Common Lisp. In Common Lisp, "character" is a distinguished data type satisfying the predicate **CL:CHARACTERP**. In Interlisp, a "character" is a single-character symbol, not distinguishable from the type symbol (litatom). Interlisp also uses a more efficient object termed "character code", which is indistinguishable from the type integer.

Interlisp functions that take as an argument a "character" or "character code" do not in general accept Common Lisp characters. Similarly, an Interlisp "character" or "character code" is not acceptable to a Common Lisp function that operates on characters. However, since Common Lisp characters are a distinguished datatype, Interlisp string-manipulation functions are willing to accept them any place that a "string or symbol" is acceptable; the character object is treated as a single-character string.

To convert an Interlisp character code *n* to a Common Lisp character, evaluate (**CL:CODE-CHAR** *n*). To convert a Common Lisp character to an Interlisp character code, evaluate (**CL:CHAR-CODE** *n*). For character literals, where in Interlisp one would write (**CHARCODE** *x*), to get the equivalent Common Lisp character one writes #*x*. In this syntax, *x* can be any character or string acceptable to **CHARCODE**; e.g., #\GREEK-A.

Chapter 4 Strings

(I:4.1)

Interlisp strings are a subtype of Common Lisp strings. The functions in this chapter accept Common Lisp strings, and produce strings that can be passed to Common Lisp string manipulation functions.

Chapter 5 Arrays

Interlisp arrays and Common Lisp arrays are disjoint data types. Interlisp arrays are not acceptable arguments to Common Lisp array functions, and vice versa. There are no functions that convert between the two kinds of arrays.

Chapter 6 Hash Arrays

Interlisp hash arrays and Common Lisp hash tables are the same data type, so Interlisp and Common Lisp hash array functions may be freely intermixed. However, some of the arguments are different; e.g., the order of arguments to the map functions in **IL:MAPHASH** and **CL:MAPHASH** differ. The extra functionality of specifying your own hashing function is available only from Interlisp **HASHARRAY**, not **CL:MAKE-HASH-TABLE** , though the latter does supply the three built-in types specified by *Common Lisp, the Language*.

Chapter 7 Numbers and Arithmetic Functions

(I:7.2)

The addition of Common Lisp data structures within the Lisp environment means that there are some invariants which used to be true for anything in the environment that are no longer true.

For example, in Interlisp, there were two kinds of numbers: integer and floating. With Common Lisp, there are additional kinds of numbers, namely ratios and complex numbers, both of which satisfy the Interlisp predicate **NUMBERP**. Thus, **NUMBERP** is no longer the simple union of **FIXP** and **FLOATP**. It used to be that a program containing

```
(if (NUMBERP X)
  then (if (FIXP X)
           then ...assume X is an integer ...
         else ...can assume X is floating point...))
```

would be correct in Interlisp. However, this is no longer true; this program will not deal correctly with ratios or complex numbers, which are **NUMBERP** but neither **FIXP** nor **FLOATP**.

Section 7.2 Integer Arithmetic

When typing to a *new* Interlisp Executive, the input syntax for integers of radix other than 8 or 10 has been changed to match that of Common Lisp. Use # instead of |, e.g., #b10101 is the new syntax for binary numbers, #x1A90 for hexadecimal, etc. Suffix Q is still recognized as specifying octal radix, but you can also use Common Lisp's #o syntax.

(I:7.4)

In the Lyric release, the FASL machinery would handle some positive literals incorrectly, reading them back as negative numbers. The numbers handled incorrectly were those numbers x greater than 2**31-1 for which (mod (integer-length x) 8) was zero. The Medley release fixes this situation. Any files containing such numbers should be recompiled.

Chapter 10 Function Definition, Manipulation, and Evaluation

Section 10.1 Function Types

All Interlisp **NLAMBDA**s appear to be macros from Common Lisp's point of view. This is discussed at greater length in *Common Lisp Impementation Notes*, Chapter 8, Macros**.**

Section 10.6 Macros

(**EXPANDMACRO** *EXP QUIETFLG — —*) [Function]

EXPANDMACRO only works on Interlisp macros, those appearing on the MACRO, BYTEMACRO or DMACRO properties of symbols. Use **CL:MACROEXPAND-1** to expand Common Lisp macros and those Interlisp macros that are visible to the Common Lisp compiler and interpreter.

Section 10.6.1 DEFMACRO

(I:10.24)

Common Lisp does not permit a symbol to simultaneously name a function and a macro. In Lyric, this restriction also applies to Interlisp macros defined by **DEFMACRO**. That is, evaluating **DEFMACRO** for a symbol automatically removes any function definition for the symbol. Thus, if your purpose for using a macro is to make a function compile in a special way, you should instead use the new form **XCL:DEFOPTIMIZER**, which affects only compilation. The *Xerox Common Lisp Implementation Notes* describe **XCL:DEFOPTIMIZER**.

Interlisp DMACRO properties have typically been used for implementation-specific optimizations. They are not subject to the above restriction on function definition. However, if a symbol has both a function definition and a DMACRO property, the Lisp compiler assumes that the DMACRO was intended as an optimizer for the old Interlisp compiler and ignores it.

Chapter 11 Stack Functions

Section 11.1 The Spaghetti Stack

Stack pointers now print in the form

`#<Stackp` address/framename>.

Some restrictions were placed on spaghetti stack manipulations in order to integrate reasonably with Common Lisp's **CL:CATCH** and **CL:THROW**. In Lyric, it is an error to return to the same frame twice, or to return to a frame that has been unwound through. This means, for example, that if you save a stack pointer to one of your ancestor frames, then perform a **CL:THROW** or **RETFROM** that returns "around" that frame, i.e., to an ancestor of that frame, then

the stack pointer is no longer valid, and any attempt to use it signals an error "Stack Pointer has been released". It is also an error to attempt to return to a frame in a different process, using **RETFROM**, **RETTO**, etc.

The existence of spaghetti stacks raises the issue of under what circumstances the cleanup forms of **CL:UNWIND-PROTECT** are performed. In Lisp, **CL:THROW** always runs the cleanup forms of any **CL:UNWIND-PROTECT** it passes. Thanks to the integration of **CL:UNWIND-PROTECT** with **RESETLST** and the other Interlisp context-saving functions, **CL:THROW** also runs the cleanup forms of any **RESETLST** it passes. The Interlisp control transfer constructs **RETFROM**, **RETTO**, **RETEVAL** and **RETAPPLY** also run the cleanup forms in the analogous case, viz., when returning to a direct ancestor of the current frame. This is a significant improvement over prior releases, where **RETFROM** never ran any cleanup forms at all.

In the case of **RETFROM**, etc, returning to a non-ancestor, the cleanup forms are run for any frames that are being abandoned as a result of transferring control to the other stack control chain. However, this should not be relied on, as the frames would not be abandoned at that time if someone else happened to retain a pointer to the caller's control chain, but subsequently never returned to the frame held by the pointer. Cleanup forms are *not* run for frames abandoned when a stack pointer is released, either explicitly or by being garbage-collected. Cleanup forms are also not run for frames abandoned because of a control transfer via **ENVEVAL** or **ENVAPPLY**. Callers of **ENVEVAL** or **ENVAPPLY** should consider whether their intent would be served as well by **RETEVAL** or **RETAPPLY**, which *do* run cleanup forms in most cases.

Chapter 12 Miscellaneous

Section 12.4 System Version Information

All the functions listed on page 12.12 in the *Interlisp-D Reference Manual* have had their symbols moved to the LISP (CL) package. They are *not* shared with the INTERLISP package and any references to them in your code will need to be qualified i.e., CL:*name*.

Section 12.8 Pattern Matching

Pattern matching is no longer a standard part of the environment. The functionality for Pattern matching can be found in the Lisp Library Module called MATCH.

[This page intentionally left blank]