24. System Construction

24.1 System Construction Concepts

24.1.1 Loading

To **load** a *file* is to treat its contents as *code* and *execute* that *code*. The *file* may contain *source code* or *compiled code*.

A *file* containing *source code* is called a *source file*. *Loading* a *source file* is accomplished essentially by sequentially *reading*[2] the *forms* in the file, *evaluating* each immediately after it is *read*.

A *file* containing *compiled code* is called a *compiled file*. *Loading* a *compiled file* is similar to *loading* a *source file*, except that the *file* does not contain text but rather an *implementation-dependent* representation of pre-digested *expressions* created by the *compiler*. Often, a *compiled file* can be *loaded* more quickly than a *source file*. See Section 3.2 (Compilation).

The way in which a *source file* is distinguished from a *compiled file* is *implementation-dependent*.

24.1.2 Features

A *feature* is an aspect or attribute of Common Lisp, of the *implementation*, or of the *environment*. A *feature* is identified by a *symbol*.

A *feature* is said to be *present* in a *Lisp image* if and only if the *symbol* naming it is an *element* of the *list* held by the *variable* ***features***, which is called the *features list*.

24.1.2.1 Feature Expressions

Boolean combinations of *features*, called *feature expressions*, are used by the `#+` and `#-` *reader macros* in order to direct conditional *reading* of *expressions* by the *Lisp reader*.

The rules for interpreting a *feature expression* are as follows:

feature
: If a *symbol* naming a *feature* is used as a *feature expression*, the *feature expression* succeeds if that *feature* is *present*; otherwise it fails.

`(not` *feature-conditional*`)`
: A **not** *feature expression* succeeds if its argument *feature-conditional* fails; otherwise, it succeeds.

`(and` *feature-conditional*`*)`
: An **and** *feature expression* succeeds if all of its argument *feature-conditionals* succeed; otherwise, it fails.

`(or` *feature-conditional*`*)`
: An **or** *feature expression* succeeds if any of its argument *feature-conditionals* succeed; otherwise, it fails.

24.1.2.1.1 Examples of Feature Expressions

For example, suppose that in *implementation* A, the *features* `spice` and `perq` are *present*, but the *feature* `lispm` is not *present*; in *implementation* B, the feature `lispm` is *present*, but the *features* `spice` and `perq` are not *present*; and in *implementation* C, none of the features `spice`, *lispm*, or `perq` are *present*. The next figure shows some sample *expressions*, and how they would be *read*[2] in these *implementations*.

```
(cons #+spice "Spice" #-spice "Lispm" x)

in implementation A ...  (CONS "Spice" X)
in implementation B ...  (CONS "Lispm" X)
in implementation C ...  (CONS "Lispm" X)

(cons #+spice "Spice" #+LispM "Lispm" x)

in implementation A ...  (CONS "Spice" X)
in implementation B ...  (CONS "Lispm" X)
in implementation C ...  (CONS X)

(setq a '(1 2 #+perq 43 #+(not perq) 27))

in implementation A ...  (SETQ A '(1 2 43))
in implementation B ...  (SETQ A '(1 2 27))
in implementation C ...  (SETQ A '(1 2 27))

(let ((a 3) #+(or spice lispm) (b 3)) (foo a))

in implementation A ...  (LET ((A 3) (B 3)) (FOO A))
in implementation B ...  (LET ((A 3) (B 3)) (FOO A))
in implementation C ...  (LET ((A 3)) (FOO A))

(cons #+Lispm "#+Spice" #+Spice "foo" #-(or Lispm Spice) 7 x)

in implementation A ...  (CONS "foo" X)
in implementation B ...  (CONS "#+Spice" X)
in implementation C ...  (CONS 7 X)
```