30. Initializations

There are a number of programs and facilities in the Lisp Machine that require that "initialization routines" be run either when the facility is first loaded, or when the system is booted, or both. These initialization routines may set up data structures, start processes running, open network connections, and so on.

An initialization that needs to be done once, when a file is loaded, can be done simply by putting the Lisp forms to do it in that file; when the file is loaded the forms will be evaluated. However, some initializations need to be done each time the system is booted, and some initializations depend on several files having been loaded before they can work.

The system provides a consistent scheme for managing these initializations. Rather than having a magic function that runs when the system is started and knows everything that needs to be initialized, each thing that needs initialization contains its own initialization routine. The system keeps track of all the initializations through a set of functions and conventions, and executes all the initialization routines when necessary. The system also avoids re-executing initializations if a program file is loaded again after it has already been loaded and initialized.

There is something called an *initialization list*. This is a symbol whose value is an ordered list of *initializations*. Each initialization has a name, a form to be evaluated, a flag saying whether the form has yet been evaluated, and the source file of the initialization, if any. When the time comes, initializations are evaluated in the order that they were added to the list. The name is a string and lies in the car of an initialization; thus assoc may be used on initialization lists. All initialization lists also have a si:initialization-list property of t. This is mainly for internal use.

add-initialization *name form* &optional *list-of-keywords initialization-list-name*

Adds an initialization called *name* with the form *form* to the initialization list specified either by *initialization-list-name* or by keyword. If the initialization list already contains an initialization called *name*, change its form to *form*.

initialization-list-name, if specified, is a symbol that has as its value the initialization list. If it is unbound, it is initialized (!) to nil, and is given a si:initialization-list property of t. If a keyword specifies an initialization list, *initialization-list-name* is ignored and should not be specified.

The keywords allowed in *list-of-keywords* are of two kinds. These specify what initialization list to use:

:cold	Use the standard cold-boot list (see below).
:warm	Use the standard warm-boot list (see below). This is the default.
:before-cold	Use the standard before-disk-save list (see below).
:once	Use the once-only list (see below).
:system	Use the system list (see below).

:login Use the login list (see below).

:logout Use the logout list (see below).

:site Use the site list (see below).

:site-option Use the site-option list (see below).

:full-gc Use the full-gc list (see below).

These specify when to evaluate *form*:

:normal Only place the form on the list. Do not evaluate it until the time comes to do this kind of initialization. This is the default unless :system or :once is specified.

:now Evaluate the form now as well as adding it to the list.

:first Evaluate the form now if it is not flagged as having been evaluated before. This is the default if :system or :once is specified.

:redo Do not evaluate the form now, but set the flag to nil even if the initialization is already in the list and flagged t.

Actually, the keywords are compared with string-equal and may be in any package. If both kinds of keywords are used, the list keyword should come *before* the when keyword in *list-of-keywords*; otherwise the list keyword may override the when keyword.

The add-initialization function keeps each list ordered so that initializations added first are at the front of the list. Therefore, by controlling the order of execution of the additions, you can control explicit dependencies on order of initialization. Typically, the order of additions is controlled by the loading order of files. The system list (see below) is the most critically ordered of the pre-defined lists.

The add-initialization keywords that specify an initialization list are defined by a variable; you can add new keywords to it.

si:initialization-keywords *Variable*

Each element on this list defines the keyword for one initialization list. Each element is a list of two or three elements. The first is the keyword symbol that names the initialization list. The second is a special variable, whose value is the initialization list itself. The third, if present, is a symbol defining the default "time" at which initializations added to this list should be evaluated; it should be si:normal, si:now, si:first, or si:redo. This third element just acts as a default; if the list of keywords passed to add-initialization contains one of the keywords normal, now, first, or redo, it will override this default. If the third element is not present, it is as if the third element were si:normal.

delete-initialization *name* &optional *keywords initialization-list-name*

Removes the specified initialization from the specified initialization list. Keywords may be any of the list options allowed by add-initialization.

initializations *initialization-list-name* &optional *redo-flag flag-value*

Perform the initializations in the specified list. *redo-flag* controls whether initializations that have already been performed are re-performed; nil means no, non-nil is yes, and the default is nil. *flag-value* is the value to be bashed into the flag slot of an entry. If it is unspecified, it defaults to *t*, meaning that the system should remember that the initialization has been done. The reason that there is no convenient way for you to specify one of the specially-known-about lists is that you shouldn't be calling initializations on them.

reset-initializations *initialization-list-name*

Bashes the flag of all entries in the specified list to nil, thereby causing them to get rerun the next time the function initializations is called on the initialization list.

30.1 System Initialization Lists

The special initialization lists that are known about by the above functions allow you to have your subsystems initialized at various critical times without modifying any system code to know about your particular subsystems. This also allows only a subset of all possible subsystems to be loaded without necessitating either modifying system code (such as lisp-reinitialize) or such kludgy methods as using fboundp to check whether or not something is loaded.

The :once initialization list is used for initializations that need to be done only once when the subsystem is loaded and must never be done again. For example, there are some databases that need to be initialized the first time the subsystem is loaded, but should not be reinitialized every time a new version of the software is loaded into a currently running system. This list is for that purpose. The initializations function is never run over it; its "when" keyword defaults to :first and so the form is normally only evaluated at load-time, and only if it has not been evaluated before. The :once initialization list serves a similar purpose to the defvar special form (see page 19), which sets a variable only if it is unbound.

The :system initialization list is for things that need to be done before other initializations stand any chance of working. Initializing the process and window systems, the file system, and the ChaosNet NCP falls in this category. The initializations on this list are run every time the machine is cold or warm booted, as well as when the subsystem is loaded unless explicitly overridden by a :normal option in the keywords list. In general, the system list should not be touched by user subsystems, though there may be cases when it is necessary to do so.

The :cold initialization list is used for things that must be run once at cold-boot time. The initializations on this list are run after the ones on :system but before the ones on the :warm list. They are run only once, but are reset by disk-save thus giving the appearance of being run only at cold-boot time.

The :warm initialization list is used for things which must be run every time the machine is booted, including warm boots. The function that prints the greeting, for example, is on this list. Unlike the :cold list, the :warm list initializations are run regardless of their flags.

The :before-cold initialization list is a variant of the :cold list. These initializations are run before the world is saved out by disk-save. Thus they happen essentially at cold boot time, but only once when the world is saved, not each time it is started up.

The :site initialization list is run every time a new site table and host table are loaded by update-site-configuration-info. The keyword :site implies :now unless overridden.

The :site-option initialization list is run every time the site options may have changed; that is, when a new site tables are loaded or after a cold boot (to see the per-machine options of the machine being booted on).

The :full-gc initialization list is run by the function si:full-gc just before garbage collecting. Initializations might be put on this list to discard pointers to bulky objects, or to turn copy lists into cdr-coded form so that they will remain permanently localized.

The :login and :logout lists are run by the login and logout functions (see page 648) respectively. Note that disk-save calls logout. Also note that often people don't call logout; they often just cold-boot the machine.

User programs are free to create their own initialization lists to be run at their own times. Some system programs, such as the editor, have their own initialization list for their own purposes.