33. 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.

It is easy to perform an action when a file is loaded: simply place an expression to perform the action in the file. But this causes the action to be repeated if the file is loaded a second time, and often that should not be done. Also, this does not provide a way to cause actions to be taken at other times, such as when the system is booted or when a garbage collection is started.

The *initialization list* facility serves these needs. An initialization list is a symbol whose value is a list of *initializations*, put on by various programs, all to be performed when a certain event (such as a cold boot) happens. When the event occurs, the system function in charge of handling the event (si:lisp-reinitialize, for cold boot) executes all the initializations on the appropriate list, in the order they are present on the list.

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. The name is a string or a symbol and lies in the car of an initialization; thus **assoc** may be used on initialization lists to find particular initializations.

System and user files place initializations on initialization lists using the function **add-initialization**. The name of the initialization is specified so that the system can distinguish between adding a new initialization and repeating or changing the definition of an initialization already known: if there is already an initialization with the specified name, this is a new definition of the same initialization. One can specify that the initialization be executed immediately if it is new but not if it is repeated.

User programs are free to create their own initialization lists to be run at their own times.

33.1 System Initialization Lists

There are several initialization lists built into the system. Each one is invoked by the system at a specific time, such as immediately after a cold boot, or during **disk-save**. A user program can put initializations on these lists to cause actions to be taken at those times as the program needs. This avoids the need to modify system functions such as **lisp-reinitialize** or **disk-save** in order to make them interact properly with the user program.

The system initialization lists are generally identified by keywords rather than by their actual names. We name them here by their keywords. In each case, the actual initialization list symbol is in the si package, and its name is the conventional keyword followed by '-initialization-list'. Thus, for :cold, there is si:cold-initialization-list. This is just a convention.

Unless otherwise specified, an initialization added to a system list is not run when it is added, only when the appropriate event happens. A few system lists are exceptions and also run each initialization when it is added. Such exceptions are noted explicitly.

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. When a new initialization is added to this list, it is executed immediately; but when an initialization is redefined, it is not executed again.

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.

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. For cold boots, the :cold initializations are done before the :warm ones.

The :system initialization list is like the :warm list but its initializations are run *before* those of the :cold list. These are generally very fundamental system initializations that must be done before the :cold or :warm initializations can work. Initializing the process and window systems, the file system, and the Chaosnet NCP falls in this category. By default, a new initialization added to this list is run immediately also. 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 :before-cold initialization list is used for things to be run 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. By default, adding an initialization to this list runs the initialization immediately, even if the initialization is not new.

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). By default, adding an initialization to this list runs the initialization immediately, even if the initialization is not new.

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 :after-flip initialization list is run after every garbage collection flip, at the beginning of scavenging. These initializations can force various objects to be copied into new space near each other simply by referencing them all consecutively.

The :after-full-gc initialization list is run by the function si:full-gc just after a flip is done, but before scavenging.

The :login and :logout lists are run by the **login** and **logout** functions (see page 801) respectively. Note that **disk-save** calls **logout**. Also note that often people don't call **logout**; they just cold-boot the machine.

33.2 Programming Initializations

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*, it is redefined to execute *form*.

initialization-list-name, if specified, is a symbol that has as its value the initialization list. If it is void, 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. Most specify the initialization list to use; a list of such keywords makes up most of the previous section. Aside from them, four other keywords are allowed, which specify when to evaluate *form*. They are called the *when-keywords*. Here is what they mean:

: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, :once, :site or :site-option is specified.
: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.
:now	Evaluate the form now unconditionally as well as adding it to the list.
: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 is the most critically ordered of the predefined 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 overrides 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*

Performs 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. This is done by the system when it is appropriate.

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.