

# Allegro CL Startup

This document contains the following sections:

[1.0 The Allegro directory](#)

[2.0 Allegro CL Executables: alisp, alisp8, mlisp, mlisp8, allegro, allegro-ansi](#)

[3.0 The Allegro CL license file](#)

[4.0 Starting Allegro CL](#)

[4.1 Starting on UNIX machines](#)

[4.1.1 Starting on UNIX using a shell script](#)

[4.2 Starting on Windows machines](#)

[4.2.1 Starting Allegro CL on Windows as a Console App](#)

[4.3 The executable, the image, and additional files](#)

[4.4 The executable and image names](#)

[4.5 Argument defaults](#)

[5.0 Command line arguments](#)

[6.0 Files Lisp must find to start up and files it may need later](#)

[6.1 Files Lisp needs to start up 1: .so \(or dll\) files built with image](#)

[6.2 Files Lisp needs to start up 2: the Allegro directory](#)

[7.0 The start-up message](#)

[8.0 Running Lisp as a subprocess of Emacs](#)

[8.1 Starting Lisp as a subprocess of Emacs the first time](#)

[8.2 Starting Lisp within Emacs after the first time](#)

[8.3 What if the Emacs-Lisp interface does not start?](#)

[8.4 Using the IDE with Emacs](#)

[9.0 Starting Lisp from a shell](#)

[10.0 Start-up problems](#)

[11.0 How to exit Lisp](#)

[11.1 How to exit for sure](#)

[12.0 What Lisp does when it starts up](#)

[13.0 Initialization and the sys:siteinit.cl and \[.\]clinit.cl files](#)

[13.1 Errors in an initialization file](#)

[13.2 No top-level commands in initialization files](#)

[13.3 Cannot \(effectively\) set a variable bound by load](#)

[13.4 Starting Allegro Composer from .clinit.cl](#)

[14.0 Setting global variables in initialization files](#)

[14.1 Where are the bindings defined?](#)

[14.2 Many bindings are to specific values, not to the variables' actual values](#)

[14.3 How to set the value so a listener sees it?](#)

[14.4 A sample initialization file](#)

[15.0 After Lisp starts up](#)

[15.1 The initial prompt](#)

[15.2 Errors](#)

[15.3 What if the system seems to hang?](#)

[15.4 Enough C-c's \(on Unix\) will always interrupt](#)

[15.5 The Allegro Icon on the system tray will interrupt on Windows](#)

[15.6 Help while running Lisp](#)

[15.6.1 The package on startup](#)

[16.0 Files that may be looked for on startup and after startup](#)

## 1.0 The Allegro directory

The *Allegro directory* is the directory where Allegro CL was installed. In it is the executable (see [Section 2.0 Allegro CL Executables: alisp, alisp8, mlisp, mlisp8, allegro, allegro-ansi](#) below for executable names) and one or more standard images (with extension *dsl* and names the same as the executable names) and other files and subdirectories which may be needed to run Allegro CL. If you follow the standard installation procedure, the Allegro directory on Windows machines is *C:\Program Files\acl62* and on Unix is */usr/acl62*.

## 2.0 Allegro CL Executables: alisp, alisp8, mlisp, mlisp8, allegro, allegro-ansi

Allegro CL comes in at least four varieties, depending on the internal representation of strings and characters (whether 16 or 8 bits per character), and the case mode (whether ANSI or modern). Therefore, your distribution will have at least the following executables and corresponding image (*dsl*) files. Note too that on Windows, the executable files have extension *.exe*, while on Unix, the executables have no extension.

**Note for Trial edition users:** On UNIX platforms, only one executable/image is supplied. On all UNIX platforms it is *alisp/alisp.dsl* (ANSI supporting international characters). A *readme* in the Allegro directory tells you how to build other types of images (but only 16-bit images can be built). On Windows, none of the four executables/images listed next are supplied. Only *allegro-ansi.exe/allegro-ansi.dsl*, described after the bulleted list.

- *mlisp* (*mlisp.exe* on Windows): Common Lisp, modern mode (case sensitive lower, supports international 16-bit characters). Some Allegro CL-based products work better in this mode, AllegroServe and Allegro ORBLink being two.
- *mlisp8* (*mlisp8.exe* on Windows): Common Lisp, modern mode (case sensitive lower, uses 8-bit characters and cannot support international character sets).
- *alisp* (*alisp.exe* on Windows): Common Lisp, ANSI mode (case insensitive upper, supports international 16-bit characters). Not all Allegro CL-based products work well in this mode -- modern languages such as C, C++, and XML are case sensitive, and supporting interfaces to these languages in a case insensitive Lisp is difficult. This is a copy of *mlisp* (*mlisp.exe* on Windows), renamed simply to start the image of the same name (and extension *dxl*).
- *alisp8* (*alisp8.exe* on Windows): Common Lisp, ANSI mode (case insensitive upper, uses 8-bit characters and cannot support international character sets). This is a copy of *mlisp8* (*mlisp8.exe* on Windows), renamed simply to start the image of the same name (and extension *dxl*).

On Windows, there are two additional executables: *allegro.exe* and *allegro-ansi.exe*. Both are copies of *mlisp.exe*, and so support 16-bit characters. They are associated with images (same name, extension *dxl*) that include the Common Graphics/IDE code and, when invoked, start the IDE automatically. (**Trial users:** only one executable/image is supplied: the *allegro-ansi.exe/allegro-ansi.dxl* (ANSI with the IDE supporting international characters). A *readme.txt* in the Allegro directory tells you how to build other types of images.) See *How to create an 8-bit image which starts the IDE* in *cgide.htm* for information on building an 8-bit image that includes the IDE and invokes it on startup.

Allegro CL has always supported modern (case-sensitive, lowercase preferred) mode, but, starting with release 6.0, a modern image and executable are supplied. We believe the benefits of using modern mode outweigh the slight inconveniences of porting case-insensitive code to it.

Allegro CL has also for some time supported international character sets, but with a special, add-on version called International Allegro CL. Starting in release 6.0, standard Allegro CL supports 16-bit characters (and strings) and does not support 8-bit characters. Because of automatic compatibility routines when loading text files, most users who do not use international character sets will not notice the change. However, some may (for example, users with applications that manipulate very large ASCII strings) and those users may wish to use the 8-bit versions. Note that a fasl file containing a string constant compiled in the standard (16-bit) Lisp cannot be loaded into the 8-bit version, but fasl files with string constants compiled in the 8-bit version can be loaded into the standard version.

Because of the multiplicity of executable names, we will refer to the executable as *mlisp* in the document unless there is some reason to refer to a different version.

## Special executables for AllegroStore users on some HP's

On HP's running HP/UX 11, there are special executables for AllegroStore only: *aslisp* is the 16-bit character version with AllegroStore and *aslisp8* is the 8-bit character version.

## 3.0 The Allegro CL license file

Allegro CL uses a license file, *devel.lic* in the Allegro directory. This file *must* be present and have proper contents in order for Allegro CL to start. If the file is not present or does not have the proper contents, Lisp fails to start with a message like the following example from the Windows version, which appears in a pop-up dialog. On UNIX, the message (with a different path) appears in a shell window or in an Emacs buffer.

```
Missing or invalid license file (C:\Program Files\ACL62\devel.lic).
Lisp cannot start up.
```

Note that the name of the license must be *devel.lic*. A valid file with a different name will not work (but the fix is easy -- copy that file to *devel.lic* in the Allegro directory).

This license file is not included on the distribution CD (or as part of the trial download). As described in *installation.htm*, information on obtaining license files is sent to Professional and Enterprise customers. Trial users obtain a license file by following the instructions on the Franz Inc. website ([www.franz.com](http://www.franz.com)). If you do not have a license file or your license file is not valid, please contact Franz Inc. See *Where to report bugs and send questions* in *introduction.htm*.

Along with allowing Allegro CL to start up, the license indicates which add-on products and utilities are licensed, and prevents access to unlicensed products and utilities. If you believe you are licensed for a product or utility but the license file prevents you from using it, again, contact Franz Inc. See *Where to report bugs and send questions* in *introduction.htm*.

Here is what a license file looks like. It is a text file, but some of the text (the actual license parts) is encrypted. This is a bogus file, of course, with the encrypted portions much modified (and much shortened). We provide it so you can see what a license file looks like. This license file is used internally at Franz Inc. (as it says). It also has an expiration date. Trial versions will have an expiration date. Licensed versions (that is, to paying customers) have no expiration date.

```
;; This file contains the licenses necessary to
;; execute Allegro CL and add-on products. Preceding
;; each license there is a comment describing the
;; contents of the encrypted block of text. Included
;; in the encrypted text is information identifying
;; the licensee. If you have you questions or comments,
;; please contact us at supportfranz.com (+1
;; 510-452-2000, Franz Inc., 555 12th St.,
;; Suite 1450, Oakland, CA 94607, USA).
;;
```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ACL version: nil
;; ACL type: enterprise
;; Architecture: <all>
;; Licensed to: franz inc
;;             bugs@franz.com
;; Expiration date: 2001-11-15 00:00:00
;; Features: partners runtime generate-application
;;   aodbc enterprise-ssl enterprise-jlinker
(:lisp
 "laGau77GHGd5LNEjcoj8qIKCuojz6GDVU8bwWToAYmnsGwsRGlGoVpkGTKM
5q8GMUCHLEFTOUT1HaMbeVfCWtboF6ds4PY0SNSE3bcc
kFl4zpHoWoVCitFch5RMpKNobLW8Crkh6+6LjQ4HyE5scS0909iu2;jacoihqw9h
FsQidHv0XouM=")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ACL version: nil
;; Architecture: <all>
;; Licensed to: franz inc
;; Expiration date: 2001-11-15 00:00:00
(:dynamic-runtime
 "la984rkjkdckjGGHGd5LPlimbMAcevJB/gKaVUxAmIMBiExRVBNF
NadITOWMUCHLEFTOUTDkNK7S2JHpmHl6nI9Ko=")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ACL version: nil
;; Architecture: <all>
;; Licensed to: franz inc
;; Expiration date: 2002-11-15 00:00:00
(:clim
 "laGakkjHGd5LauHe9kOSnujYE8FJ+nLedxTtOB4wwKr8xD5V+TEWc
WcMBICaaMUCHLEFTOUTjvFckoXbw=")

;; END OF FILE

```

---

## 4.0 Starting Allegro CL

---

### 4.1 Starting on UNIX machines

The recommended way to start Lisp on UNIX machines is as a subprocess of Emacs (Xemacs or GNU Emacs). However, Lisp may be started from a shell. The disadvantage of starting Lisp from a shell is that the editing and other features of the Emacs-Lisp interface are not available. If you use Allegro Composer, you must run Lisp as a subprocess of Emacs.

The command for starting in a shell (assuming the Allegro directory is in your PATH) is:

```
> mlisp [-I <image path and name>.dxl] [other args]
```

See [Section 5.0 Command line arguments](#) for information on starting under emacs and for information on command-line arguments including the -I argument. See [Section 2.0 Allegro CL Executables: alisp, alisp8, mlisp, mlisp8, allegro, allegro-ansi](#) for the names of executables other than *mlisp*.

## 4.1.1 Starting on UNIX using a shell script

The UNIX hack of having a file that starts with

```
#! <program> <args>
```

works with Allegro CL on some UNIX platforms (it does not work on HP-UX or FreeBSD -- see [HP and FreeBSD note below](#)). Here is the form:

```
#! <lisp-executable> { -#D | -#C | #T | -#! }
```

Where:

- <lisp-executable> is the name of the Allegro CL executable file. Note: <lisp-executable> cannot itself be a shell script: it must be one of the actual executable files, such as *mlisp*.

Further, on most platforms, the executable should be in a directory which is in the PATH environment variable (despite the fact that the full pathname of the file is specified in the script).

The reason the directory needs to be in PATH is that the Allegro CL executable needs to find itself on startup (to do things like finding its symbol table). However, most versions of UNIX seem to discard the directory information in the script after finding the specified executable, so Lisp itself has only the executable's filename to use to find itself. (If the executable's location is in PATH, it is able to find itself.) Note that certain versions of UNIX, such as Linux with kernels greater than 2.4 and Solaris do keep the specified executable's location information and on those platforms, it does not seem to be necessary to have the location included in PATH.

Initialization files are never read when Lisp is started with a script. So Lisp is started as if the command-line argument `-qq` was specified. See [Section 5.0 Command line arguments](#) for information on command-line arguments.

- `-#D` is used for debugging (a backtrace is printed when an error occurs).
- `-#C` is used to have the script compiled.
- `-#T` is like `-#C`, in that the script is compiled, but the compiled script is placed in `/tmp` rather than the directory containing the script, and so the user need not have write permission in that directory.
- `-#!` just loads the script.

Scripts run using `#!` that signal an error will exit with a non-zero exit status.

Here's an example. Put this in a file `hello_world.cl`. We assume `PATH` includes `/usr/local/acl62/`:

```
#!/usr/local/acl62/mlisp -#!
(format t "Hello World!~%")
```

Then,

```
% chmod 755 hello_world.cl
% ./hello_world.cl
Hello World!
%
```

Because different versions of UNIX handle shell scripts like these differently, this method of starting Allegro CL may not work (it is known not to work on HP-UX and FreeBSD, as noted below). Please test the simple example before trying a more complex one.

## fasl files can be appended to scripts

The following transcript show this capability.

```
gemini% cat foo.cl
(in-package :user)

(format t "foo!~%")
gemini% mlisp-7.0
International Allegro CL [master]
7.0 [Linux (x86)] (Apr 23, 2004 22:46)
Copyright (C) 1985-2004, Franz Inc., Berkeley, CA, USA. All Rights
Reserved.
```

```

;; Optimization settings: safety 1, space 1, speed 1, debug 2.
;; For a complete description of all compiler switches given the current
;; optimization settings evaluate (explain-compiler-settings).
;;---
;; Current reader case mode: :case-sensitive-lower
cl-user(1): :cf foo.cl
;;; Compiling file foo.cl
;;; Writing fasl file foo.fasl
;;; Fasl write complete
cl-user(2): :exit
; Exiting
gemini% echo '#! usr/local/acl/7.0/mlisp -#!' > foo.sh
gemini% cat foo.fasl >> foo.sh
gemini% chmod 755 foo.sh
gemini% ./foo.sh
foo!
gemini%

```

## HP and FreeBSD Note

When running a shell script on HP-UX, `argv[0]` is not the Lisp specified in the script, but is the script file specified on the command-line. Thus, HP-UX is preserving the original name as the value of `argv[0]`. Other UNIX systems set `argv[0]` to the interpreter given in the script.

Because of this difference, the example in this section will not work on HP-UX platforms.

The script also does not work on FreeBSD, which includes `./hello_world.cl` as an (unknown) command-line argument.

---

## 4.2 Starting on Windows machines

There is an Allegro 7.0 menu item on the Windows Start menu (under Programs). Choosing it displays a submenu, whose items include Modern ACL images and ANSI ACL Images. These two display submenus with various choices, the first submenu of modern (case-sensitive) images and the second of ANSI (case-insensitive) images. On each, the item `'Allegro CL (w IDE)'` starts Allegro CL with the IDE and starts the IDE automatically.

To start using Emacs, see [Section 8.0 Running Lisp as a subprocess of Emacs](#). See [Section 5.0 Command line arguments](#) for information on command-line arguments, including the `-I` argument (which specifies the image to use). See [Section 2.0 Allegro CL Executables: `alisp`, `alisp8`, `mlisp`, `mlisp8`, `allegro`, `allegro-ansi`](#) for the names of executables other than `mlisp.exe`.

If you want to start without using a menu item, you can choose Run from the start menu, then specify the executable file, image file, and any additional needed or desired arguments (described in [Section 5.0 Command line arguments](#)), like this:

```
> <Allegro directory>\mlisp.exe [-I <image path and name>.dxl]
[other args]
```

---

## 4.2.1 Starting Allegro CL on Windows as a Console App

You can start Allegro CL on Windows as a Console Application by using the executable *build.exe* rather than *mlisp.exe*. *build.exe* is a WIN32 console application, whereas *mlisp.exe* is a WIN32 Windows application. If you want to run a command-line oriented version of Allegro CL, use *build.exe*.

Note that there are issues with multiprocessing and *build.exe*. *build.exe* sometimes cannot tell if input is available or if a process that is waiting on input should wake up. For this reason, we recommend you do not use *build.exe* with multiprocessing applications.

Note further that *build.exe* does not accept the arguments that begin with +. Command-line arguments are described in [Section 5.0 Command line arguments](#).

---

## 4.3 The executable, the image, and additional files

Allegro CL requires at least two files: an executable, typical extension *exe*, and an image file, typical extension *dxl*. The executable is small and can be used with any *dxl* file (from the same release and on the same platform). The *dxl* file is typically much larger and contains information on the Lisp environment. (Note that in earlier releases on Unix, the image file was executable.)

You run the executable specifying a *dxl* file as an argument (identified by the *-I* flag). If no *dxl* file is specified, the executable looks for a *dxl* file in the same directory as the executable (typically, the Allegro directory) and with the same name. Therefore, if the executable is */usr/acl62/mlisp* and no *dxl* file is specified, */usr/acl62/mlisp.dxl* will be looked for and used if it is present. If it is not present, the startup will fail (and a *-I* argument must be specified in that case).

---

## 4.4 The executable and image names

See [Section 2.0 Allegro CL Executables: \*alisp\*, \*alisp8\*, \*mlisp\*, \*mlisp8\*, \*allegro\*, \*allegro-ansi\*](#) for the names of all executables supplied with Allegro CL 7.0. The name of the standard executable is *mlisp* on Unix and *mlisp.exe* on Windows. Any executable can be copied to have any other name and doing so is useful if you want it automatically associated with an image file with that name. You should leave the executable in the Allegro directory (where it was installed off the CD) because its location serves as a pointer to the locations of files needed during execution.

The location of the executable is set when the software is installed. Typically, your system administrator will know where Allegro CL was installed (if you did not install the software yourself). The image files have extensions *dxl*. One or more are supplied with the distribution and are in the same directory as the executable. Images can also be created with **dumplisp** and **build-lisp-image**. Their location and name depends on the arguments specified to those functions. It is not necessary for the executable and the image to be in the same directory. The `-I` command-line argument specifies the image file to use and its value can contain directory information as well as the name of the image file.

In this manual, we typically assume that the name of the Allegro CL executable is *mlisp* or *mlisp.exe*, as the example is on Unix or on Windows, and the image *mlisp.dxl* or, on Windows when using the IDE, *allegro.dxl* or *allegro-ansi.dxl*. When the platform is not explicit, either *mlisp* or *mlisp.exe* may be mentioned and readers should mentally add or remove *.exe* as they are working on Windows or on Unix. We also assume that the executable is located in a directory that is in your PATH environment variable (that means the image can be started by typing the image name without its path). You, of course, should use the executable name actually used on your system and should include its path or not as appropriate.

## 4.5 Argument defaults

Values for command line arguments can be specified as a resource. On Unix, this is done with the file *lisprc* in the same directory as the executable. On Windows, the arguments are stored in the executable. This feature is designed for applications. Using resources is discouraged for general use.

It has significant limitations; in particular, the arguments specified in the resource are concatenated with those specified directly on the command-line with the resource arguments first. That is, the resource + arguments (Windows only) are processed, then the command-line + (Windows only) arguments, then the resource – arguments, and finally the command-line – arguments. Since the first `-I` argument encountered is used and others are ignored, you cannot override the resource `-I` argument if one is specified. (Actually, on Unix, you can move the *lisprc* file but on Windows, there is no such workaround.)

It is because of these limitations that using resources is discouraged for general use. We mention them here because if resources are used, users may not get the behavior they expect and may not easily figure out why. Resources are described in the document *delivery.htm*, in the section on *Resources*. See that document for information on determining whether resources are being used and for information on specifying them for applications.

## 5.0 Command line arguments

A command line argument is specified after the executable name when starting Lisp from a UNIX shell, in response to a question when starting Lisp as a subprocess of Emacs, and in the Run dialog when starting under Windows. You may specify any of the predefined arguments described next and any other arguments you want. The arguments specified may affect the startup procedure but in any case, all specified arguments are available when the image is running using functions like **command-line-arguments**.

Note that it is not necessary to specify any arguments when running Allegro CL so long as the dxl file has the same name and is in the same directory as the executable file or has its name and location stored in the executable file. If the dxl file has a different name or is in a different directory and is not specified in the executable file, it must be identified by a `-I` argument.

Command-line arguments are prefixed with a plus (+) or a minus (-). Those prefixed with a plus (+) are for Windows only. Those prefixed with a minus (-) are for all platforms.

The general form for command line arguments on Windows is (it should be all on one line but we have broken the line for ease of reading):

```
mlisp.exe [acl-arguments starting with +] [-I dxl-file]
          [other acl-arguments starting with -]
          [-- application-arguments]
```

We have displayed the `-I` argument separately from the other `-` arguments because of its special importance.

On UNIX, the general form of the command-line, shown here when starting from a shell, is

```
% mlisp [-:] [-I dxl-file] [acl-arguments starting with -]
          [-- application-arguments]
```

`-:` (controlling how the Allegro CL shared object file is to be looked for) must be specified first, if supplied. When starting in Emacs, you are asked for a 'Lisp Image (dxl) file' and also for 'Image arguments (separated by spaces)'. The image (value of the `-I` argument) should be specified to the first question and any remaining command-line arguments in answer to the second.

*acl-arguments* affect how Lisp is run. *application-arguments* have no direct effect on Lisp but are available once Lisp starts up. A double minus (`--`) separates the *acl-arguments* from the *application-arguments*. Everything before the double minus is interpreted as an *acl-argument*; everything after is interpreted as an *application-argument*. Therefore, the same value can appear in either place (or in both places) without

ambiguity. If an argument before the `--` is unrecognized by Allegro CL, a warning is printed.

*application-arguments* (and *acl-arguments* as well) are available after Lisp starts up with the function **command-line-arguments** and its relatives. We will not say more about *application-arguments* here. See the discussion of command-line arguments in *Accessing command-line arguments* in *os-interface.htm*. See also **with-command-line-arguments**, which is a macro that can be used to process command-line arguments (and can be useful for processing application command-line arguments).

Now to *acl-arguments*. The following tables show the arguments and describe their effects. The first table describes the arguments starting with a `+`, which apply to Windows only. The second table applies to all platforms. Note that (on Windows) arguments starting with a `+` must precede all arguments starting with a `-`.

Argument	Effect
<code>+c</code>	Start Allegro CL without a console window. (Normally, there is a console window to read/write standard I/O. If you close the console window, the Lisp will be killed.) Note that there will not be an icon in the system tray regardless of whether <code>+R</code> or <code>+RR</code> are specified when there is no console. (Having the console minimized with <code>+cm</code> , non-activated with <code>+cn</code> , or hidden with <code>+cx</code> does not affect whether there is a system tray icon.)
<code>+cm</code>	Start the console window in a minimized state.
<code>+cn</code>	Start the console window, but don't activate the console window. This allows Lisp to be started so as to not interfere with the currently selected window.
<code>+cx</code>	Start the console in a hidden state. Double-clicking on the tray icon will make the console visible. See also the right-click menu on the tray icon.
<code>+cc</code>	Causes any earlier <code>+c</code> , <code>+cm</code> , <code>+cn</code> , or <code>+cx</code> argument, including those specified as a resource, to be ignored. (The purpose of this argument is to override such arguments specified as resources, see the section on <i>Resources</i> in <i>delivery.htm</i> .) Any of <code>+c</code> , <code>+cm</code> , <code>+cn</code> , or <code>+cx</code> can be specified and will be effective after <code>+cc</code> .
<code>+p</code>	Preserve the console window. Without this switch the console window goes away when Lisp exits with a zero exit status. If Allegro CL is exiting unexpectedly you can use this switch to keep the window around to find out what it printed before it died. This is the opposite of the <code>+M</code> argument.
<code>+R</code>	Don't put the Allegro CL icon in the tray.
<code>+RR</code>	Do put the Allegro CL icon in the tray. This argument is useful when <code>+R</code> (described just above) is in a resource command line and you wish to override it. It is not necessary to specify this argument unless <code>+R</code> is specified in a resource. See <i>Resources</i> in <i>delivery.htm</i> . Note that specifying <code>+c</code> results in no icon in the system tray regardless of whether this argument is specified.

+s <i>scriptfile</i>	Standard input is initially bound to this file. When the file is completely read, standard input reverts to the console window, if there is one. (This argument is also accepted on UNIX, where it is interpreted as -s, described in the next table, thus allowing uniform scripts for Windows and UNIX.)
+M	When this argument is present the console window does not need to be closed manually when Lisp exits with a non-zero exit status. This is the opposite of the +p argument.
+m	When this argument is present, Lisp starts the lisp up in a minimized state.
+d <i>dll-name</i>	Use the given Allegro CL dll instead of the default <i>acl&lt;version&gt;.dll</i> .
+t <i>title</i>	Sets the title of the console to <i>title</i> .
+B	Use no splash screen (a window that appears very quickly to let you know that Lisp is starting).
+b <i>text</i>	Make the splash screen display <i>text</i> instead of the default bitmap. <i>text</i> just appears in a gray box.
+Bt	Display the splash screen for three seconds as the Lisp application is starting up. The splash screen is stored in the image file. See the description of <b>set-splash-bitmap</b> .
+Bp	Display the splash screen indefinitely, until either the user clicks on it or the application closes it programmatically by using one of the Common Graphics functions <b>kill-splash-screen</b> or <b>kill-splash-screen-when-ready</b> . The splash screen is stored in the image file. See the description of <b>set-splash-bitmap</b> .
+Cx	Disable the Close button on the Console Window. This argument and the next (+Tx) make it difficult for users to close the running Lisp application accidentally. This is useful for applications that are NT services running in the background performing some useful task (e.g. providing an NFS server).
+N <i>appname</i>	Change the name of the application from Lisp to <i>appname</i> in items in the system tray menu, so the items will be <b>Interrupt appname</b> and <b>Exit appname</b> rather than <b>Interrupt Lisp</b> and <b>Exit Lisp</b> .
+Tx	Disable the "Exit Lisp" menu item on the system tray menu. This argument and the previous (+Cx) make it difficult for users to close the running Lisp application accidentally. This is useful for applications that are NT services running in the background performing some useful task (e.g. providing an NFS server). If an application name other than Lisp was specified by the +N argument, the disabled item will say "Exit <appname>".
+Ti	Remove the "Interrupt Lisp" menu item on the system tray menu. (The item says "Exit <appname>" is an application name other than Lisp is specified with the +N argument.)

+<number>	Specify with <i>number</i> the maximum size in bytes of the console edit control (i.e. the amount of text that can appear in the console). For example, the value +35000 would specify 35000 bytes. The default size 100,000 on Windows. The size can be specified. It must be at least 1000 but should be substantially higher in most situations. On platforms where the size can be specified, it can be changed after startup with <b>console-control</b> .
-----------	---

Arguments starting with a -; these apply to all platforms:

Argument	Effect	See notes
- :	<p>UNIX only. Causes the Lisp shared library (<b>libacl&lt;version&gt;.so</b> or <b>libacl&lt;version&gt;.sl</b>) to be searched for in a system-dependent way. On Solaris 2 this means using the environment variable <b>LD_LIBRARY_PATH</b>; other systems might use other ways.</p> <p>This argument must precede all others. (This argument is processed by the executable before the image file is loaded.)</p>	
-I <i>image-file</i>	<p>Specifies the image file to use. This image must have been created with <b>dumplisp</b> or its relative <b>build-lisp-image</b> (or <b>generate-application</b>).</p> <p>The filename of the image file <i>must</i> have an extension (the standard extension is <i>dxl</i> but any extension will do). If the extension is <i>dxl</i>, the image can be specified without <i>.dxl</i>, so these are equivalent:</p> <pre>mlisp -I foo.dxl</pre> <pre>mlisp -I foo</pre> <p>If the extension is something other than <i>dxl</i>, it must be specified:</p> <pre>mlisp -I foo.xxx</pre> <p><b>mlisp.exe</b> (Windows) or <b>mlisp</b> (Unix) handles the -I command line argument specially: if <b>mlisp.exe/mlisp</b> is started without an image file (i.e., no -I argument), then it will first look for an image file with the name of the</p>	

executable and the type *dxl* in the current directory, then in the same directory as the executable file. (On Windows, these two directories are often the same but need not be, particularly if you are starting from a DOS prompt. On Unix, the current directory is the result of **pwd** typed to the prompt used to start Allegro CL.)

That is, if you start *c:\x\y\z\mlisp.exe* it will look for the image file *c:\x\y\z\mlisp.dxl*. If it fails to find that image file it will prompt for an image file. Note: you can change the name of **mlisp.exe/mlisp** if you are delivering an application. If you change **mlisp.exe/mlisp** to *myapp.exe/myapp* then when *myapp.exe/myapp* starts it will look for *myapp.dxl*.

If more than one **-I** argument is specified, the first (leftmost) is used and the remainder are ignored. (This means that if a **-I** argument is specified as a resource, it cannot be overridden. See the section on *Resources* in *delivery.htm* for information on resources.)

-q

Read working directory *.clinit.cl* or *clinit.cl* and *sys:siteinit.cl*, but do not read *~/.clinit.cl* or *~/clinit.cl* unless *~* is also the working directory. On Unix, the working directory is specified to Emacs, or, the current directory if starting in a shell. On Windows, the current directory is usually the directory containing the executable (.exe) file that is invoked, but may be something different, such as a directory specified in *Start In* field of a shortcut. On Unix *~* refers to the user's home directory. On Windows, the home directory is the directory which is the value of the HOME environment variable, or C:\ if HOME is unset.

*sys:siteinit.cl* is hardwired in the system. The actual names of the other initialization files are in a list which is the value of *\*init-file-names\**, whose initial value is (*.clinit.cl clinit.cl*) – that is *clinit.cl* with and without a leading dot.

Do not also specify **-qq**.

-qq

Do not read *sys:siteinit.cl* or any initialization file. Do not also specify **-q**.

-C <i>file</i>	evaluates ( <code>compile-file file</code> )	
-d <i>file</i>	Causes dribbling to <i>file</i> .	
-H	This argument is no longer supported. In previous releases, it tried to set the Allegro directory location (the translation of the <code>sys:</code> logical host, the location where Lisp library files will be looked for), but this did not work consistently. The translation of <code>sys:</code> is the Allegro directory location is the directory where the executable ( <code>mlisp.exe/mlisp</code> ) is located.	
-kill	Evaluates ( <code>exit 0</code> ). That is, Lisp exits. Presumably you have other arguments which do things (like <code>-C</code> compiling a file) earlier in the list of command line arguments. Thus  <code>mlisp.exe -C foo.cl -kill</code>  will start Allegro CL, compile <i>foo.cl</i> , and exit.	
-L <i>file</i>	Evaluates ( <code>load (string file)</code> ). Only one file per <code>-L</code> argument but as many <code>-L</code> arguments as desired can be specified. The <code>-L</code> arguments are processed from left to right.	1, 3
-locale <i>locale-name</i>	Sets the initial locale (the initial value of <code>*locale*</code> ) to the locale named by <i>locale-name</i> , which must be the name of a locale available on the machine. See <i>The initial locale when Allegro CL starts up</i> and <i>External formats and locales</i> , both in <i>iacl.htm</i> for details.	1, 3
-Q	Currently unused. In 5.0.1 and earlier, this argument suppressed printing the name of the image and library file but now those filenames are not printed in any case. This argument is kept for backward compatibility.	

-s	<p>[UNIX only] A UNIX version of the +s argument for Windows described above. When specified with a file as an argument, cause the forms in file to be read, evaluated and their results printed. Unlike +s Windows, -s does not have to be before any arguments that start with `'. If the file specified does not have an (exit 0) at the end, <code>-s file -kill</code> is a good way to make the Lisp exit.</p> <p>This argument is superior to using a UNIX pipe because it allows interactive debugging of the Lisp process, which cannot be done when a pipe is used.</p> <p>You can use +s instead of -s (+s will be interpreted on UNIX as -s). This means you do not have to conditionalize between Windows and UNIX.</p>	
-W	Evaluates (setq *break-on-warnings* t)	
-e <i>form</i>	Evaluates <i>form</i> . Please see <a href="#">Further description of the -e and -ee command-line arguments</a> below.	1, 2, 3
-ee <i>form</i>	Evaluates <i>form</i> after replacing escaped characters represented as %hh when hh is a hex number. The character whose code is hh is used. Please see <a href="#">Further description of the -e and -ee command-line arguments</a> below.	1, 2, 3
-batch	Run in batch mode: input comes from standard input, exit when an EOF is read. Exit on any error (but print a backtrace to *error-output* if -backtrace-on-error also specified).	4
-backtrace-on-error	Print a complete backtrace (as printed by <b>:zoom</b> :all t :count t) whenever an error occurs.	6
-compat-crlf	When specified, #\return #\linefeed translates to '13 10' (as it did in release 5.0.1) instead of '13 13 10' (as it does now without this option). See <i>#newline Discussion</i> in <i>iacl.htm</i> for details. Only users with code from Allegro CL 5.0/5.0.1 for Windows should even consider using this option, and most of those do not need it.	5

-!	Please see <a href="#">Special Note on the -! command-line argument</a> below. This argument should only be used in unusual debugging situations and should never be used in ordinary situations.	
-project <i>project-lpr-file</i>	[Windows when running the Integrated Development Environment only] have the project specified by the indicated project <i>.lpr</i> file be the current project when the IDE is started. See <i>About IDE startup</i> in <i>cgide.htm</i> .	5

## Table notes:

1. All arguments, including **-e form**, **-ee form**, and **-L file** arguments, are read and processed in order. Reading of one **-e/-ee** form can depend on previous forms having been evaluated and previous **-L** files having been loaded. For example, if the package `mypack` is defined in *file.fasl*, the following will not fail:

```
-L file.fasl -e (mypack::myfun)
```

because the `mypack` package has been defined when the reader encounters **`mypack::myfun`** (since *file.fasl* has been loaded).

2. When Lisp is started as a subprocess of Emacs, the system adds the arguments `-e (start-emacs-lisp-interface t)` as the first arguments after the image name. Your own **arguments are not affected by this since multiple -e's are permitted, but the emacs-supplied arguments will appear along with the image name under the 'command' column in the information printed by the Unix ps command.**
3. It may not be possible to recover from an error signaled during the evaluation of a **-e** form or the loading of a **-L** file (indeed, Lisp may exit without giving you the opportunity to recover). Even if you are presented with a prompt to type to, many top-level commands may not be available. Restarting the image with the form corrected is the recommended action.
4. When running in batch mode, if an error occurs, Lisp exits with error code 1. If an EOF is encountered, Lisp exits with error code 0.
5. This argument (`-backtrace-on-error`) is designed for use with `-batch`. It can be used with a non-batch image, but a backtrace is printed whenever an error occurs and most users find this annoying. A backtrace can be printed. The file `<allegro directory>/src/autozoom.cl` defines a macro with `autozoom-and-exit` which can be wrapped around code where programmers want to force a backtrace (and the backtrace can be directed to any stream, not just `*error-output*`). That macro is much more appropriate for non-batch programs.

## Further description of the -e and -ee command-line arguments

Both **-e** and **-ee** take a form as a companion argument, The problem with the **-e** argument is that certain characters, such as the space character, might be part of the form or it might be a delimiter indicating the start of a new command-line argument. The system has difficulty distinguishing those cases.

The traditional way to get around that problem is to escape such characters, typically with a **\** (backslash), or to wrap the companion argument in single or double quotes. Suppose, for example, you want to evaluate the form `(setq foo 10)` at startup. In answer to the 'Image arguments' question asked by **fi:common-lisp** (the Emacs function to start Allegro CL), you could put:

```
-e (setq\ foo\ 10)
```

And starting Lisp as a shell prompt, you might do

```
% mlisp -e '(setq foo 10)'
```

These solutions generally work, but the escaping becomes more complicated with more complex forms. Further, when writing shell scripts to start Allegro CL, the shell processing may undo the escapes at the wrong time so when the argument is actually read, the characters that need escapes no longer have them.

The **-ee** argument is designed to get around that problem. When a **-ee form** argument is processed, the companion argument is scanned for character triplets of the form **%hh**, when **hh** is a pair of Hex numbers (0-9abcdef). Every **%hh** triplet is replaced by the character whose **char-code** is **hh**. Then the companion argument is processed.

Frequently used codes include:

- #( -- code %28
- #) -- code %29
- #: -- code %3a
- #\space -- code %20
- #\% -- code %25

The `(setq foo 10)` argument could then be passed

```
-ee %28setq%20foo%2010%29
```

or

```
% mlisp -ee '%28setq%20foo%2010%29'
```

Thus the notation allows Lisp forms to be passed as a single OS token unaffected by command line parsing or substitutions.

The function **make-escaped-string** takes a Lisp form (as a string) and returns a properly converted string where the characters that need to be escaped are replaced with the appropriate %hh triple.

Note that only characters with **char-codes** less than 256 can be represented.

### Special note on the -! command-line argument

If this argument is specified, it must appear *first* among the command-line arguments. This argument is typically used when using a C-based debugger like **gdb**. It should never be used in ordinary circumstances because it causes certain kinds of error which are typically not fatal to become fatal errors.

The -! command-line argument is designed to assist in using an external debugger. The primary effect of this switch is to prevent lisp from catching illegal memory references and turning them into calls to the segmentation violation handler. Because these bad references are not caught, the exception generated by the operating system will be passed to whatever program is controlling lisp (usually a C Debugger). Note that if there is no program controlling Lisp, then the operating system will put up a dialog box announcing the exception and Lisp will not be able to continue. A secondary effect is to start up in a more verbose mode.

## 6.0 Files Lisp must find to start up and files it may need later

The next series of headings describe files Lisp needs to start up. A complete list of such files can be found in [Section 16.0 Files that may be looked for on startup and after startup](#) below. Here we give an abbreviated description directed at users who are running standard installed images or images dumped with standard installed images but run in situ (i.e. the dumped image is not moved to another machine). Programmers who do intend to move dumped images to different machines (or who are trying to start such an image but having difficulty) and application programmers wishing to deliver an application should refer to [Section 16.0 Files that may be looked for on startup and after startup](#) or to *delivery.htm*.

### 6.1 Files Lisp needs to start up 1: .so (or dll) files built with image

Like any UNIX or Windows program, Lisp may have shared object (.so or .sl files on Unix, .dll files on

Windows) files mapped into the image at build time. This is done with the *user-shared-libraries* argument to **build-lisp-image**.

On Unix, these files are looked for in the Allegro directory, the directory where the executable file is located (unless a `-H` command-line argument specifies another location).

On Windows, the following method is used:

With both implicit and explicit linking, Windows first searches the set of pre-installed DLLs such as the performance library (KERNEL32.DLL) and the security library (USER32.DLL). Windows then searches for the DLLs in the following sequence:

1. The directory where the executable module for the current process is located. (this is the likely location and corresponds to Unix behavior.)
2. The current directory.
3. The Windows system directory. The `GetSystemDirectory` function retrieves the path of this directory.
4. The Windows directory. The `GetWindowsDirectory` function retrieves the path of this directory.
5. The directories listed in the `PATH` environment variable.

Note that the `LIBPATH` environment variable is not used.

## 6.2 Files Lisp needs to start up 2: the Allegro directory

The Allegro directory is a directory that contains files Lisp may need to start up and while it is running. Because it needs files like the *.pll* file, Lisp may fail on startup if it cannot find this directory. The Allegro directory also refers to the directory where Allegro CL was installed off the CD. We used the same name because they are typically the same directory.

The *.pll* filename is stored in the image, either as just a filename, just a filename and type, or a filename (with or without a type) with directory information. If directory information is supplied, the *pll* file is looked for in that location only (using default type *pll* if no type is specified), relative to the current directory if the directory information is relative. No other location is searched in that case. If the *pll* file information in the image file is just a filename or filename and type, it is searched for in the current directory and then in the Allegro directory, and then, on Windows only, in the Windows systems directory. See **pll-file**.

Unless the `-H` command-line argument is specified, the Allegro directory is assumed to be the directory containing the executable (the file, initially *mlisp.exe* on Windows and *mlisp* on Unix). Files that may be needed for running may be in that directory or in the *code/* subdirectory. If `-H` is specified, the files must be

in the directory specified. The structure of that directory should mimic the structure of the Allegro directory (that is, it should have a *code/* subdirectory and files should be placed in the same relative locations as they are in the Allegro directory).

(In the 4.3.x releases on Unix, this directory was found using an environment variable named `ALLEGRO_CL_HOME`. That was necessary because the image file was executable and could be located anywhere, including a user's home directory. Now that the image is separate from the executable, the location of the executable can serve as a reference point for finding other needed files. The image can still be anywhere, including users' home directories. Earlier releases on Windows such as 3.0.x always used the separate executable and image model and always used the executable location to find needed files.)

Once Lisp finds the Allegro directory, it sets the translation of the logical host `sys:` to be that directory. That directory contains most of the files Lisp needs to start up. It may happen that the directory identified as the Allegro directory exists but is incorrect (that is, does not actually contain the files Lisp needs). Lisp will fail to start if it needs a *.pll* file but cannot find it in the current directory or identified Allegro directory. Other missing files may cause problems as well, but Lisp usually gets some way into the startup process. Dealing with an existing but misidentified Allegro directory is much like dealing with a non-existent one: find where the Allegro directory really is and communicate this information to Lisp.

## 7.0 The start-up message

When Lisp starts up, it prints the banner, which looks something like (this example is from a Sun):

```
Allegro CL 7.0 [SPARC; R1]
Copyright (C) 1985-2004, Franz Inc., Berkeley, CA, USA. All Rights
Reserved.
```

The exact version number, the machine type, and the date and time will all likely differ in your Lisp.

Then (technically as part of the banner) information on who this version of Allegro CL is licensed to is printed.

The `sys:siteinit.cl` and the initialization files (`.clinit.cl` or `clinit.cl`) are read next (see [below](#)) and then the Emacs-Lisp interface is started (if Lisp was started as a subprocess of Emacs). Then the start-up message is printed. The default start-up message says (the actual `Current reader case mode type` depends on the type of image, this example is from a modern image):

```
;; Optimization settings: safety 1, space 1, speed 1, debug 2
;; For a complete description of all compiler switches given the
;; current optimization settings evaluate (EXPLAIN-COMPILER-SETTINGS).
```

```
;; --
;; Current reader case mode: :case-sensitive-lower
```

(The number of lines may differ.)

Start-up messages are printed by the generic function **print-startup-info**. Exactly what is printed is controlled by the variable `*print-startup-message*`.

---



---

## 8.0 Running Lisp as a subprocess of Emacs

Your Emacs should have loaded all the Emacs-Lisp interface definitions. See *What should be in your .emacs file* in *eli.htm* for details.

If you are using the Integrated Development Environment on Windows, see [Section 8.4 Using the IDE with Emacs](#) below. Otherwise, see [Section 8.1 Starting Lisp as a subprocess of Emacs the first time](#) just below.

---

## 8.1 Starting Lisp as a subprocess of Emacs the first time

This is the standard way to start a Lisp image. When an image is started in this way, all of the Emacs-Lisp interface will be available, making it easier to use Lisp. The Emacs-Lisp interface is documented in *eli.htm*.

Allegro CL works with Xemacs or GNU Emacs. To avoid repetition, we simply say Emacs to refer to either product. The Release Notes may contain the exact version numbers and other information about these products.

To start Allegro CL within Emacs, enter the following:

```
M-x fi:common-lisp
```

(M-x means depress the `x' key while the Meta key is down or press the Escape key and then press the x key.) Emacs will ask 6 questions:

1. **Buffer:** what the buffer in which Lisp will run should be named. The default is the value of the Emacs variable `fi:common-lisp-buffer-name`, whose value is typically `*common-lisp*`.
2. **Host:** the name of the machine where Lisp will run. The default is the value of the Emacs variable `fi:common-lisp-host`.
3. **Process directory:** the directory in which to run Lisp (if you were running from the shell, this is the

directory in which Lisp is started). The default is the value of the Lisp variable `fi:common-lisp-directory`. This directory will be the one returned by **current-directory** and the initial value of `*default-pathname-defaults*` after Lisp starts.

4. **Executable Image name:** the name of the Lisp executable. This is a small file. Its name has no extension on UNIX and has extension (type) *exe* on Windows. Various executable files are shipped with the Allegro CL distribution. The file itself is not typically created or modified at customer sites.

There are two distinct executable files and often several copies of each. One executable file uses 16-bit characters (copies are named, for example, *mlisp/mlisp.exe* and *alisp/alisp.exe*). The other uses 8-bit characters (copies are named, for example, *mlisp8/mlisp8.exe* and *alisp8/alisp8.exe*). On Windows, there is also *allegro.exe* and *allegro-ansi.exe*. The filename is used to determine the image name (so *mlisp/mlisp.exe* looks for the image file *mlisp.dxl*, and so on).

The default is the value of the Emacs variable `fi:common-lisp-image-name`. (The name is historic: in earlier versions the executable file and the image file were the same. Now they are different. The variable `fi:common-lisp-image-file` has the image file as its value.)

5. **Image:** the name of the Lisp image (.dxl) file. The default is the value of the Emacs variable `fi:common-lisp-image-file`. (Note that, for historic reasons, the emacs variable `fi:common-lisp-image-name` names the executable file.) If this argument is not specified, its value is assumed to be the file in the same directory as the executable image name with the same name but with extension .dxl. Thus, if Executable Image Name is *mlisp*, the file *mlisp.dxl* in the same directory of *mlisp* will be looked for if this argument is unspecified.
6. **Image arguments:** the command line arguments described above. Since they are separated by spaces, you must use appropriate quotation marks or backslashes if your arguments contains spaces (it may take several tries to get it right). The default is the value of the Emacs variable `fi:common-lisp-image-arguments`. Enter nothing if you want no arguments.

All questions are asked in the Emacs minibuffer. Once you have answered all the questions, Lisp is started according to your specifications.

You can write your own Emacs function to call `fi:common-lisp` with your desired values (and thus avoid having to input values). See *Functions and variables for Interacting with a CL subprocess* in *eli.htm* for examples of such functions.

**Warning:** you must exit Lisp before quitting Emacs or killing the Common Lisp buffer. Doing either without exiting Lisp may leave the Lisp process running. Starting Lisp again from within Emacs after you have killed a `*common-lisp*` buffer without exiting from Lisp may result in Emacs being connected to the wrong running image.

---

## 8.2 Starting Lisp within Emacs after the first time

Emacs remembers your answers to the questions asked by `fi:common-lisp` so if you evaluate `fi:common-lisp` again (after killing off the first Lisp image for whatever reason), Emacs will use the old answers without asking again. If you want the questions asked again (because, e.g. you want to run on a different host or with different initial arguments), use the universal argument (typically `C-u`) before `M-x fi:common-lisp`. Thus:

```
C-u M-x fi:common-lisp
```

The defaults will be your previous set of answers.

**Warning:** if the cursor is at the end of the `*common-lisp*` buffer and superkeys are enabled, `C-u` will be interpreted as ``delete back to the last CL prompt'` rather than as universal argument. If Lisp is not running in the buffer and you enter `C-u` at the end of the buffer, Emacs will print

```
Wrong type of argument: processp, nil
```

in the minibuffer. To get `C-u` interpreted as universal argument in this case, go up a line (with `previous-line`, typically `C-p`) where `C-u` will then be interpreted as universal argument:

```
C-p C-u M-x fi:common-lisp
```

## 8.3 What if the Emacs-Lisp interface does not start?

When Lisp is started with Emacs, the Emacs-Lisp interface should start up automatically. The system will print:

```
;; Starting socket daemon and emacs-lisp interface...
```

in the `*common-lisp*` buffer and

```
Trying to start connection...done
```

in the minibuffer. If both these appear, the connection has been started. If they do not appear or if the ``done'` does not appear, the interface may not have started. If the Emacs-Lisp interface does not start-up, you can start it up by evaluating the form:

```
(start-emacs-lisp-interface)
```

It is not an error to call this function when the interface has in fact started. If you do so, however, something similar to the following will appear in the Lisp listener:

```
^A1026 771990 :upper NIL 1^A
```

(You will likely see different numbers and perhaps :lower rather than :upper. This message can be ignored.)

---

## 8.4 Using the IDE with Emacs

Windows only. The Integrated Development Environment (IDE) is a collection of graphical tools for building a graphical interface to an application developed in Allegro CL. It provides editors and editing capability. However, users familiar with Emacs may prefer to use Emacs.

The complication of using the IDE and Emacs is that all evaluation of IDE-related code that receives events (mouse input of any sort and keyboard input to a dialog) must be done by the IDE. This is because all IDE-related events must happen within the IDE thread (see the document *multiprocessing.htm*). Emacs runs in a separate thread. IDE-associated events cannot be initiated by the Emacs thread (it is an error to do so). Thus, function definitions etc. can be done within emacs. But code that displays dialogs and handles events must be run within the IDE thread using IDE tools.

When using the IDE, specify *allegro.dxl* or *allegro-ansi.dxl* as the Image file.

---

## 9.0 Starting Lisp from a shell

You can start Lisp from a shell prompt or a DOS prompt. Assuming on UNIX the name of the Lisp executable is *mlisp* and that the image is located in a directory specified in the PATH environment variable, the following will start Lisp (the command is all on one line; we break it for readability and indicate the break with a \):

```
% mlisp [-I <image>.dxl] [acl-arguments starting with -] \  
  [-- user-supplied-arguments]
```

Assuming on Windows the name of the Lisp executable is *mlisp.exe* and that the image is located in a directory specified in the PATH environment variable, the following will start Lisp (the command is all on one line – we break it for readability and indicate the break with a \):

```
% mlisp.exe [-I <image>.dxl] [acl-arguments starting with -] \  
  [-- user-supplied-arguments]
```

If *mlisp/mlisp.exe* is not in your PATH, you can specify the full path in order to start Lisp.

A Lisp started from a shell on UNIX will not benefit from the Emacs-Lisp interface and many features of Allegro Composer will not work.

---

---

## 10.0 Start-up problems

Generally, there are not many problems with start-up. If there is insufficient swap space to run Lisp, a message saying so will be printed and Lisp will not start. Insufficient swap space may be a temporary problem but if you often are prevented from starting for that reason, you should consider increasing the swap space on your machine (see the operating system's System Administration manual for information on increasing swap space).

Most errors during startup cause Lisp to fail (it prints the error message). The exceptions include errors during the loading of an initialization file and failure to find a loaded .so file. Those errors produce warnings but do not prevent the startup procedure from continuing. Other errors cause failure. Usually, the cause is something added to the startup procedure (perhaps by a `-e` command line argument or an item on `*restart-actions*`). The recommended action is to try to start again without the `-e` arguments or to use `-e` arguments to set the value of `*restart-actions*` to `nil` (perhaps first making it the value of another variable which can be examined after startup succeeds).

Other potential problems include the following:

- The Emacs-Lisp interface does not start. See the information under the heading [Section 8.3 What if the Emacs-Lisp interface does not start?](#) above.
  - The Allegro directory location cannot be found or necessary files cannot be found. Finding the Allegro directory is rarely a problem since it is typically the same directory as the executable (.exe) file. (The `-H` command-line argument can be used to specify a different location and startup may fail if that location does not exist, however.) Lisp may also fail to start because needed files are not in the Allegro directory.
  - **Lisp fails on startup (HP only)**. Are you running a version for HP-UX 10.20 on an HP-UX 11.x machine? See the HP-specific information in *release-notes.htm* for information on this.
- 
- 

## 11.0 How to exit Lisp

The ANSI spec is silent on the question of how to exit Lisp. Allegro CL has several commands which will cause a Lisp session to terminate. The expression

```
(exit)
```

will cause the current Lisp image to exit.

One can also exit Lisp directly from the top level, using the top-level command **:exit**, documented in *top-level.htm*. These operators behave somewhat differently according to whether the Lisp is running multiprocessing or not. If there are multiple processes running, the user may be concerned that these processes exit cleanly (with windows cleaned up, for example).

The functions used for exiting have options which allow for clean exits in each process, but the user should be aware that if some problem occurs, user intervention may be necessary to correct it, and it may take some time for the problem to manifest itself. You may have to respond to questions (about, for example, whether certain processes should indeed be killed) before exiting completes. If you are running on Windows using the IDE (see *cgide.htm*), you will usually have to confirm the exit by clicking **Yes** on a modal dialog asking if you really want to exit Lisp, and, often, also indicate whether unsaved files should be saved.

The top level command may not exit if there are other processes running, and the function **exit** called without arguments will try to clean up all processes that are running. Users using multiprocessing should read the description of the **exit** function. (The Emacs-Lisp interface uses multiprocessing. However, processes created by that interface, or by Allegro Composer, are generally well-behaved with respect to exit behavior. Problems typically arise with user-created processes.)

## 11.1 How to exit for sure

If a simple call to (**exit**) does not cause Lisp to exit (typically because of problems with multiple processes), try the following form:

```
(excl:exit 0 :no-unwind t)
```

## 12.0 What Lisp does when it starts up

In this section, we describe the Allegro CL startup procedure. While this is usually of academic interest only, if the startup does not go as expected, the reason may be that things are done in a different order than you expect. See the file *src/aclstart.cl* (in the Allegro directory), which contains the source to the startup

code.

1. **The initial value of `*locale*` is set.** (See *The initial locale when Allegro CL starts up* in *iacl.htm* for details. Note that the `-locale` command-line argument is not examined until command-line processing is done several steps down.)
2. **All logical pathnames translations are flushed.** The translations may have been defined in the parent image (particularly if the image was created with **dumplisp** after the initial build). They are flushed because there is no guarantee that they are valid. The image will find translations anew by looking in the files in the list returned by **logical-pathname-translations-database-pathnames**. `sys:hosts.cl` is on that list. `sys:` itself translates to the Allegro directory.
3. On Windows NT/2000, the affinity (number of physical processors to be used) is set to 1. See **processor-affinity-mask**.
4. **The standard streams are bound.** The standard streams are `*terminal-io*`, `*standard-input*`, `*standard-output*`, `*error-output*`, `*trace-output*`, `*query-io*`, and `*debug-io*`. They are bound to `*initial-terminal-io*`.
5. **The Allegro CL banner is printed.** (It is a violation of the license agreement to suppress the printing of the Allegro CL banner except under certain conditions. Contact your Franz Inc. account manager if this is an issue.)
6. **Loaded `.so` (`.sl`, `.dll`) files are reloaded.** Shared object files are mapped into a running image, not built into the executable image. therefore, all such files must be reloaded on startup. Under certain circumstances, the image may not be able to find these files. If it cannot find a `.so` file, a warning is printed but no error is signaled. If Lisp gets to the first prompt without error, the missing `.so` file can usually be loaded at that time.
7. **Command-line arguments are processed.** All `-e` and `-L` arguments are processed, in order, at this time. (A `-e` argument specifies a form to evaluate. A `-L` argument specifies a file to load.)
8. **Initialization files are read.** The files are `sys:siteinit.cl`, the two user initialization files. Whether the files are read and exactly which files are read depends on the value of `*read-init-files*` and the command-line arguments. If any initialization files are read, `sys:siteinit.cl` is read first. If both user initialization files are read, the one in the home directory is read first. Errors that occur reading the files are ignored. If an error occurs in a file, the remainder of the file is ignored. User initialization files are named by the variable `*init-file-names*`. The initial value of this variable is the list `(.clinit.cl clinit.cl)`. Whatever the value, only one file (the first found processing the list in order) is read from any location.
9. **The startup messages are printed.** The function **print-startup-info** is called to print the message.
10. **Restart actions are performed.** The functions on the `*restart-actions*` list are called.
11. **The initial stack-group is set up.**
12. **ACL\_STARTUP\_HOOK is examined.** The environment variable `ACL_STARTUP_HOOK` is polled. If set, its value is read by `read-from-string` and the result is evaluated. This is a last-chance backdoor into the startup process.
13. **The restart init function is called.** If `*restart-init-function*` is non-nil, its value is assumed to be a function of no arguments and it is funcall'ed.
14. **The Lisp listener is started or the restart-app function is called.** The following form is executed:

```
(tpl:start-interactive-top-level s
```

```
(or *restart-app-function*
    #'tpl:top-level-read-eval-print-loop)
nil)
```

If `*restart-app-function*` is true, its value is assumed to be a function of no arguments which is started within a **tpl:start-interactive-top-level** form (which sets up bindings for global variables). It is assumed to provide the application top-level if there is one or do whatever the application does without user interaction. It should not return.

If `*restart-app-function*` is nil, **tpl:top-level-read-eval-print-loop** is called. It starts a Lisp listener (the Initial Lisp Listener) and presents the user with a prompt.

One thing we want to emphasize by detailing the startup sequence is that because of the order, certain things cannot affect other things. Trivially, a command-line argument can affect whether and which init files are read but an init file cannot affect command-line processing. Note the following about the startup procedure:

- The source code for the startup procedure is in `<Allegro directory>/src/aclstart.cl`. That file can be examined for more details of starting up.
- The steps 2-11 are run within a handler-case form. If an unhandled error occurs during startup, a message is printed and Lisp exits. The acts of reading the init files and reloading loaded .so files are protected against errors so problems there do not cause unhandled errors.
- Step 11 provides a backdoor hook into Lisp via an environment variable. In some cases, this hook can be used to force a startup which is failing. If, for example, `*restart-app-function*` is causing problems, but init files are not read and command-line arguments are ignored (two other ways to affect startup), starting Lisp in an environment where `ACL_STARTUP_HOOK` is `"(setq *restart-app-function* nil)"`, Lisp will start a Lisp listener instead of calling `*restart-app-function*`. We strongly advise against setting this environment variable except in an emergency.

## 13.0 Initialization and the `sys:siteinit.cl` and `[.]clinit.cl` files

When Allegro CL is first invoked it may look for and load if present several initialization files. The variable `*read-init-files*` controls whether the files are looked for at all and the command-line arguments `-q` and `-qq` also control which or whether init files are read. The files are:

- `sys:siteinit.cl`. The `sys:` component names the system directory, which is interpreted by Allegro CL to be the Allegro directory. If any initialization files are read, this one is read first.
- `~/clinit.cl` or `~/clinit.cl` (that is, with and without a leading dot). On Unix, `~` is the current user's home directory. On Windows, `~` is the value of the `HOME` environment variable, or `C:\` if `HOME` is unset. The first named file found is the only one read.

- `<current-directory>/clinit.cl` or `<current-directory>/clinit.cl` (unless the `<current directory>` is `~` and `~/.clinit.cl` or `~/clinit.cl` has been loaded). On Unix, the current directory is the working directory where Lisp is started. On Windows, the current directory is usually the directory containing the executable (.exe) file that is invoked, but may be something different, such as a directory specified in *Start In* field of a shortcut. The first named file found is the only one read (that is, only one of `clinit.cl` and `.clinit.cl` will be read even if both are present).

The value of the variable `*init-file-names*` is a list of strings naming the initialization files to look for (other than `sys:siteinit.cl`). Its initial value is `("clinit.cl" ".clinit.cl")`. We use a name with and a name without a leading dot because of the difference between handling of initial dots on Unix and on Windows. (On Unix, a leading dot means do not normally include the file in a file listing. On Windows, a leading dot is just confusing.) In a directory, the files are looked for in the order they appear in the list and the first one found (and only that one) is read.

Any valid Lisp form may be present in an initialization file. Initialization files are often used to customize your Lisp environment, by, for example, loading files or changing reader syntax. Loading of initialization files can be suppressed with an argument on the command line that initiates Lisp. If `-q` is specified on the command line, the `[.]clinit.cl` in your home directory will not be read (unless that is also the current directory). When `-q` is specified, the `sys:siteinit.cl` file and the `[.]clinit.cl` file in the current directory (even if it is also your home directory) are looked for and loaded if present.

If `-qq` is specified on the command line, no initialization file will be read.

## 13.1 Errors in an initialization file

Note that an error in an initialization file will not cause Lisp to enter a break loop, although an error message will be printed. If you wish to debug an initialization file, load it explicitly, using the `-qq` argument if necessary to suppress its initial loading. As a simple example, consider a `[.]clinit.cl` file containing the forms:

```
(setq xx (firt '(1 2 3 4)))
      (setq yy 5)
```

In the first form, `firt` is a misprint for the Common Lisp function `first`. When Lisp is initialized and the `.clinit.cl` file is read, messages similar to the following are printed:

```
Error: attempt to call `firt' which is an undefined function.
      Error loading #p"/h/dm/.clinit.cl"
```

Lisp does not enter a break level upon errors in a `[.]clinit.cl` file. Instead it aborts further processing of the file. (Thus `yy` will be unbound.) The command

```
:ld .clinit.cl
```

will, on the other hand, generate a continuable error allowing the user to correct the bad function name.

---

## 13.2 No top-level commands in initialization files

Top-level commands (prefixed by the top-level command character) cannot be used from within the initialization file, or any other file. They may only be typed to the top level. **tpl:do-command** does provide a functional equivalent of a top-level command (see the description in *top-level.htm*). Note, however, that the information necessary to successfully perform certain commands (such as `:zoom`) is not available when the initialization files are loaded and so the call may fail, in some cases with a recursive (i.e. non-recoverable) error.

---

## 13.3 Cannot (effectively) set a variable bound by load

Note that certain global variables are bound when `load` is loading a file. Therefore, setting those variables in an initialization file will not have the desired effect. (See also the discussion of setting global values in initialization files below). The following table shows what variables are bound by `load`:

Variable	Bound to
<code>*package*</code>	<code>*package*</code>
<code>*readtable*</code>	<code>*readtable*</code>
<code>*source-pathname*</code>	name of file being loaded
<code>*redefinition-warnings*</code>	<code>*redefinition-warnings*</code>
<code>*libfasl*</code>	value of <i>libfasl</i> keyword argument to <b>load</b> , if specified, or <code>*libfasl*</code>

---

## 13.4 Starting Allegro Composer from .clinit.cl

UNIX only. Simply putting `(composer:start-composer)` in your *.clinit.cl* file works but has the annoying side

effect of printing several lines of bogus warnings. Better is to use the following code.

```
(defun start-composer-from-clinit-file ()
  (let ((initial-restart-init-function *restart-init-function*))
    (cond (initial-restart-init-function
           (setf *restart-init-function*
                 #'(lambda ()
                     (composer:start-composer)
                     (setf *restart-init-function*
                           initial-restart-init-function)
                     (funcall initial-restart-init-function))))
          (t
           (setf *restart-init-function*
                 #'(lambda ()
                     (composer:start-composer)
                     (setf *restart-init-function* nil))))))))

(start-composer-from-clinit-file)
```

---



---

## 14.0 Setting global variables in initialization files

Allegro CL starts a Lisp listener by binding many standard Common Lisp special variables and also many Allegro-CL-specific special variables to appropriate values. The listener runs within the scope of these bindings. Allegro CL does this because it implements a multiprocessing extension. The bindings permit one process to set a special without another process being affected. (For example, if a process doing output sets `*print-base*` to 8 in order to print out octal numbers, the Lisp listener, running concurrently and trying to print the integer nine, will print 9 rather than 11.) Even if you are running Lisp without initiating multiprocessing, the Listener runs within the scope of the bindings.

---

### 14.1 Where are the bindings defined?

A Lisp listener gets its bindings from two alists: `*cl-default-special-bindings*` and `*default-lisp-listener-bindings*`. Here are a few lines from each alist:

```
;; From excl:*cl-default-special-bindings* (a few values only)
(*PRINT-LENGTH*) (*PRINT-LEVEL*) (*PRINT-RADIX*) (*PRINT-BASE* . 10)
(*PRINT-PRETTY* . T) (*PRINT-ESCAPE* . T)
```

```
;; From tpl:*default-lisp-listener-bindings* (a few values only)
(TOP-LEVEL:*AUTO-ZOOM* . TOP-LEVEL:*AUTO-ZOOM*)
(TOP-LEVEL::*LAST-FILE-COMPILED* . TOP-LEVEL::*LAST-FILE-COMPILED*)
```

---

## 14.2 Many bindings are to specific values, not to the variables' actual values

Notice that most of the entries in `*cl-default-special-bindings*` associate a variable with a specific value such as `t` (the value associated with `*print-pretty*` and `*print-escape*`) or `10` (the value associated with `*print-base*`) or `nil` (the value associated with `*print-length*`, `*print-level*`, and `*print-radix*` -- the `nil` isn't printed because in an alist, `(*print-escape* . nil)` and `(*print-escape*)` are the same thing). Indeed, all of the pairs shown associate a variable with a specific value but if you look at the entire list you will see some entries that are associated with the value of a variable. Most of the entries in `*default-lisp-listener-bindings*` associate variables with values of variables, but notice most of them are unexported.

The variables associated with specific values will be bound to those values when a Lisp listener is started regardless of the actual value of the variable at the time the binding is done. This means that setting (with `setf` or `setq`) the value of such a variable in an initialization file (or anywhere prior to a Lisp listener being started) cannot affect the binding in effect within the listener. To be concrete, suppose the following form is in your `.clinit.cl` file and that file is read on Lisp startup:

```
(setq *print-length* 20)
```

If you evaluate `*print-length*` in the listener that appears after startup, you get:

```
USER(1): *print-length*
NIL
```

Why? Because the listener sets the binding of `*print-length*` to `nil`, as called for in `*cl-default-special-bindings*`. The value set in the initialization file is ignored.

---

## 14.3 How to set the value so a listener sees it?

You change the alist from which the bindings are drawn so that the binding is made to the value you want. (You could also remove the entry from the alist so that the value is not bound when a listener is started, but that is not recommended.) The macro `tpl:setq-default` is designed to do exactly that. The form

```
(tpl:setq-default *print-length* 20)
```

does the following:

- It sees whether `*print-length*` is a key in the alist which is the value of `*cl-default-special-bindings*`. If it is, it replaces the value of the pair with `(sys:global-symbol-value '*print-length*)`. If it does not appear as the key in either list, it adds

```
(*print-length* . (sys:global-symbol-value '*print-length*))
```

to the `*cl-default-special-bindings*` alist.

- It sets the global symbol value of `*print-length*` to 20 as if with the form

```
(setf (sys:global-symbol-value '*print-length*) 20)
```

Now, when a Listener is started, `*print-length*` will be bound to `(sys:global-symbol-value '*print-length*)`, which, if the global symbol value has not been subsequently changed, is 20. So, the form

```
(tpl:setq-default *print-length* 20)
```

should go in the initialization file instead of `(setq *print-length* 20)`.

## 14.4 A sample initialization file

If a file `.clinit.cl` or `clinit.cl` exists in the user's home directory (C:\ on Windows) or in the working directory (the Allegro directory on Windows), it is loaded when Lisp starts up (the first one found and only the first one found is loaded from any directory). This provides a method for customizing your Lisp environment. The sample initialization file below sets several top-level variables. There is a sample `.clinit.cl` file in the distribution at *[Allegro directory]/misc/dot-clinit.cl*.

A `sys:siteinit.cl` file (which will be read by every Lisp image at your site) will typically not have this level of customization. However, the form (using **tpl:setq-default**, e.g.) is similar.

```
;;;
;;; This file contains examples of user
;;; customizations which can be done via a
;;; $HOME/.clinit.cl or C:\clinit.cl.
```

```

(format *terminal-io* "~%; Loading home ~a~@[.~a~] file.~%"
      (pathname-name *load-pathname*)
      (pathname-type *load-pathname*))

;;; Set a few top-level variables.
(tpl:setq-default top-level:*history* 50)
(tpl:setq-default top-level:*print-length* 20)
(tpl:setq-default top-level:*print-level* 5)
(tpl:setq-default top-level:*zoom-print-level* 3)
(tpl:setq-default top-level:*zoom-print-length* 3)
(tpl:setq-default top-level:*exit-on-eof* t)

;;; Display 10 frames on :zoom,
(tpl:setq-default top-level:*zoom-display* 10)
;;; and don't print anything but the current frame on
;;; :dn, :up and :find
(tpl:setq-default top-level:*auto-zoom* :current)

;;; Have the garbage collector print interesting stats.
(setf (sys:gsgc-switch :print) t)
(setf (sys:gsgc-switch :stats) t)

;;; To have all advice automatically compiled.
(tpl:setq-default *compile-advice* t)

;;; Have packages print with their shortest nickname
;;; instead of the package name.
(tpl:setq-default *print-nickname* t)

;;; Allow concise printing of shared structure.
(tpl:setq-default *print-circle* t)

;;; Only print "Compiling" messages for files, not for individual
functions,
;;; unless there is a warning or error.
(tpl:setq-default *compile-verbose* t)
(tpl:setq-default *compile-print* nil)

;;; Set up a top-level alias.
(top-level:alias ("shell" 1 :case-sensitive) (&rest args)
  "`:sh args' will execute the shell command in `args'"
  (let ((cmd
        (apply #'concatenate 'simple-string
              (mapcar
               #'(lambda (x)

```

```

                                (concatenate
'simple-string
                                (write-to-string
x :escape nil) ""))
                                args))))
  (prin1 (shell cmd))))

;;; The following makes the source file recording
;;; facility compare only the names of pathnames, for
;;; the purposes of determining when a redefinition
;;; warning should be issued.
(push #'(lambda (old new fspec type)
          (string= (pathname-name old)
                  (pathname-name new)))
      *redefinition-pathname-comparison-hook*)

;;; Use the Composer package if it is available.
(eval-when (eval compile load)
  (when (find-package :wt)
    (use-package :wt)))

```

---



---

## 15.0 After Lisp starts up

---

### 15.1 The initial prompt

After the initial messages are printed (and the Emacs-Lisp interface is started, if appropriate), the first prompt is printed. Unless you have changed the prompt, it will look like:

```
cl-user(1):
```

Here, `cl-user` specifies the `common-lisp-user` (nicknamed `user`) package (Lisp starts in that package) and the `1` indicates the command number. You can now type forms to Lisp. Note: this is the modern mode prompt. The ANSI mode prompt is `CL-USER(1):`.

---

### 15.2 Errors

In the course of running Allegro CL, you may make an error. If Allegro CL detects an error, it will go into a break loop. The prompt will indicate a break loop by a number in brackets preceding the prompt. The following script shows what happens when an error is signaled:

```
user(2): (car '(1 2))
1
user(3): (car 1)
Error: Attempt to take the car of 1 which is not a cons.
[condition type: simple-error]
[1]user(4): (car 2)

Error: Attempt to take the car of 2 which is not a cons.
[condition type: simple-error]
[2] user(5):
```

See *top-level.htm* for more information on the prompt and break levels.

## 15.3 What if the system seems to hang?

For interrupting on UNIX, see [here](#), for interrupting on Windows, see [here](#).

At times, the system may seem to `hang', that is make no response to what you have typed and seem to be doing nothing. This may be normal, since some commands take a long time to execute, or it may indicate that the system has gone into some loop or received improper input.

If you are running in a shell, a common cause is unclosed parentheses in the input line. Try adding some closing parentheses and see if that helps -- too many right parentheses will signal a warning but not an error. (If you are running under Emacs, it should be clear that parentheses are missing because the cursor will be indented since Emacs expects more information.)

If too few right parentheses is not the problem, try to regain control by hitting the Unix interrupt character (user-settable but typically C-c in the shell and C-c C-c in Emacs). Entering this character sends signal #2 (SIGINT) to the Lisp process. Allegro CL checks for a keyboard interrupt at almost every function call and at least once in each iteration of a loop compiled at proper safety, and this method will typically work. However, it will not work in two cases: (1) when Lisp is executing foreign code (if the infinite loop is in C or FORTRAN code loading into Lisp) and (2) when Lisp is executing a function compiled so that interrupt checks are suppressed (it is possible to set the compiler options so that this check is not made in user defined compiled functions, see *compiling.htm*).

## 15.4 Enough C-c's (on Unix) will always interrupt

Often, a Control-C will interrupt on Unix (C-c C-c if using Emacs). If a single Control-C does not work, after 5 SIGINTs (C-c in a shell, C-q C-c when running Lisp under Emacs) will cause Lisp to be interrupted with a non-continuable error (a single SIGINT triggers a continuable error).

Note we say `C-q C-c' when in the Emacs-Lisp interface. Typically, C-c C-c sends a SIGINT to Lisp when running in Emacs. However, that sends it through the interface, and the interface will not process things while Lisp is in an infinite loop. C-q C-c cause the SIGINT to go directly to Lisp.

Warnings will be printed after 3 and 4 interrupts have been received, telling you that 2 more (after 3) or 1 more (after 4) will cause Lisp to signal a non-continuable error.

Lisp can do this because in fact, SIGINTs are recorded whenever they occur, but Lisp does not do anything until an appropriate moment (typically, at the beginning of a function call, unless the function was compiled in such a way as to suppress that check). If 5 SIGINTs are recorded without being handled, Lisp will now interrupt for sure. It does not matter when these SIGINTs happen (in foreign or Lisp code compiled in any fashion) except garbage collections will not be interrupted.

We chose five because users often enter several interrupts to interrupt a program, and we did not want normal behavior to trigger a non-continuable error. With the warnings mentioned above, we do not believe that users will unintentionally enter five interrupts without really wanting to break into the Lisp process.

Users should be warned that it is possible for an interrupt generated by 5 SIGINTs to leave the stack in a corrupted state. When you break into Lisp in this fashion, you should immediately enter `:pop` or `:reset` to try to clear the stack. In some cases, Lisp may not be able to recover. (Almost all those cases happen when foreign code is called with the *call-direct* argument to **def-foreign-call** -- see *foreign-functions.htm*. Please note that some Franz code, particularly CLIM code, calls foreign code using *call-direct*. We have done several things to ensure Lisp is not corrupted even when the break occurs in a *call-direct*, but failure is a possibility.) However, it is likely that you will be able to recover (and recovery was typically not possible in earlier versions when an infinite loop was entered).

Users have tried (with more or less success) to break into Lisp from another shell by sending a Signal 4 or Signal 11 to the Lisp process. This can still be done (with the same success rate) but note that only SIGINTs (Signal 2) are recorded as counting to the 5 necessary to break for sure.

## 15.5 The Allegro Icon on the system tray will interrupt on Windows

The Break/Pause key will interrupt on Windows when the system is checking for events. However, when

Break/Pause does not work, look for the Allegro icon (a bust of Franz Liszt, after whom the company Franz Inc. was named), which appears in the system tray (at the lower right) of the Windows screen. Right clicking on this icon will display a menu which includes the command `Interrupt Lisp`. Choosing that command will usually cause an error to be signaled within Lisp which will return control to the user. Note that there is a small possibility that the interrupt will occur at a time when an interrupt cannot be processed and Lisp may fail.

It is usually possible to interrupt from the Console window (where you typically have a prompt that is responsive). The same menu over the Allegro icon on the system tray has a command to display the console if it is not already visible.

## 15.6 Help while running Lisp

There are a number of facilities to assist you while you are running Lisp. Some of them are listed in this section.

Some useful Common Lisp functions include **`apropos`** and **`describe`**. **`apropos`** takes a case-sensitive string (or a symbol whose print name will be used as the string) and an optional package as arguments and returns all symbols whose names contain the string. Thus, if you can't quite remember a function name, but are sure of part of it, call **`apropos`** with the part you are sure of. We recommend using the package argument, for without it `apropos` may return many symbols internal to the system or the compiler package which clutter up the output.

**`describe`** takes a Lisp object as an argument. It is especially useful when given a symbol. Included in the information it provides about the symbol are whether it has a function binding (with the formal argument list if it does), its package, whether it is internal or external in that package, whether it is bound, and its value if it is bound.

The function **`arglist`** provides the argument list of functions and macros.

The top-level commands are listed when you enter the command **`:help`** with no arguments. All top-level commands are listed, with short descriptions. A longer description of a command is printed when you enter `help` with a command name as an argument.

The function **`print-startup-info`** (as its name implies) is used for printing information when Lisp starts up. However, the information can be useful at times other than start up. The function uses the current values of variables, compiler optimization qualities, etc. to generate its message. Evaluating the following form will cause all available information to be printed:

```
(excl:print-startup-info t nil)
```

## 15.6.1 The package on startup

By default, the initial package on startup is `common-graphics-user` (nicknamed `cg-user`) when using the IDE and `common-lisp-user` (nicknamed `cl-user`) when not using the IDE. The current package is displayed in the prompt.

Getting a different package on startup is not as easy as you might think. You cannot change packages in a file (since `*package*` is bound during file loading) so you cannot use an init file (see [Section 13.0 Initialization and the `sys:siteinit.cl` and `lclinit.cl` files](#)) and the `-e` command-line argument is processed too early to be effective.

Instead, you have to modify the `*restart-init-function*` appropriately. How to do it is described in *Specifying the initial value of `*package*` in `delivery.htm`*. `delivery.htm` discusses writing your own application (where you control the values of the various global variables). The following code in your `.clinit.cl` file (see [Section 13.0 Initialization and the `sys:siteinit.cl` and `lclinit.cl` files](#)) will modify the `*restart-init-function*` while preserving whatever else it does:

```
;;-----

(defun set-init-pack-from-clinit-file ()
  (let ((initial-restart-init-function *restart-init-function*))
    (cond (initial-restart-init-function
           (setf *restart-init-function*
                 #'(lambda ()
                     (tpl:setq-default *package* (find-package :my-
package))
                     (rplacd (assoc 'tpl::*saved-package*
                                     tpl:*default-lisp-listener-
bindings*)
                                   'common-lisp:*package*)
                     (setf *restart-init-function*
                           initial-restart-init-function)
                     (funcall initial-restart-init-function))))
          (t
           (setf *restart-init-function*
                 (tpl:setq-default *package* (find-package :my-
package))
                 (rplacd (assoc 'tpl::*saved-package*
                                 tpl:*default-lisp-listener-
bindings*)
                           'common-lisp:*package*)
                 (setf *restart-init-function* nil)))))))
```

```
(set-init-pack-from-clinit-file)
```

```
;;-----
```

## 16.0 Files that may be looked for on startup and after startup

The following two tables list files that may be looked for when Allegro CL starts up and immediately afterwards.

Table 1 lists the files that Lisp may need to start up at all. Failure to find one will result in a message from the operating system or the executable itself explaining the failure. No recovery is possible.

Table 2 lists other files that Lisp may need during or after startup. Failure to find these files may cause a Lisp warning or error. Failure to find a file in some cases causes Lisp to exit.

We distinguish between system files, application files, and user files. System files are supplied as part of Allegro CL, the operating system, or other system components such as the window system; application files are those loaded into Lisp by the application programmer prior to the creation of the executable image; user files are additional runtime files belonging to the current user and referenced after the executable image is invoked.

Certain of the files in Table 1 are looked for in the Allegro directory location.

Table 1: Files required for Lisp to start up

File type	When file is required	Where file is looked for
Application loaded .so/.sl/.dll files	When specified to :user-shared-libraries (argument to <b>build-lisp-image</b> )	Allegro directory, i.e. directory where executable (exe) file is located or see <a href="#">Section 6.1 Files Lisp needs to start up 1: .so (or dll) files built with image</a>
.pll file (always a single file, e.g. acl.pll)	Image built with :pll-file non-nil.	Allegro directory location

Bundle file (typically files.bu or files.ebu)	Executable built as Presto image	Allegro directory location
---	----------------------------------	----------------------------

After all the needed files in the above table are found, the logical pathname translations for SYS: are established to refer to the Allegro directory. The ALLEGRO\_CL\_HOME environment variable, used in release 4.3 to find files, is no longer used to locate files.

Table 2: Files looked for during or after Lisp starts up

File	When file is needed	Comments
The bundle file (typically sys:files.bu or sys:files.ebu)	File needs to be autoloaded (because of a form in init file or restart function or because of system actions such as starting Lisp/Emacs interface)	None
System fasl files, e.g. sys::code;loop.fasl	As above for <i>fasl</i> files not in bundle.	Some system fasls are not included in the bundle file, and patched autoload fasl files supersede files in the bundle.
Loaded system .so files, e.g. sys:gc-cursor.so	Look in sys:.	None
Loaded application .so files	Found using LD_LIBRARY_PATH or equivalent.	None
[user-homedir]/clinit.cl [user-homedir]/clinit.cl	Loaded during initialization unless: (1) -q or -qq specified on command line; (2) image created with :read-init-file (argument to dumplisp) nil; or (3) Specified restart function does not return.  The homedir on Windows is C:\.	None
sys::siteinit.cl	Loaded during initialization unless (1) *read-init-files* is nil, or (2) -qq command-line argument specified	None

<p><i>[current-dir]/.clinit.cl</i></p> <p><i>[current-dir]/clinit.cl</i></p>	<p>Loaded during initialization unless: (1) -qq specified on command line; (2) *read-init-files* is nil or :nohome, or (3) specified restart function does not return.</p> <p>The current directory on Windows (on startup) is the directory containing the executable (usually mlisp.exe) file.</p>	<p>The names are specified in the list which is the value of *init-file-names*, whose initial value is (.clinit.cl clinit.cl). If that variable is nil, no file will be looked for.</p> <p>Only one file, the first one found looking in the order they appear in the list, will be loaded. If both <i>.clinit.cl</i> and <i>clinit.cl</i> are in a directory, only <i>.clinit.cl</i> will be loaded (assuming *init-file-names* has its initial value).</p>
<p><i>sys::hosts.cl</i></p>	<p>load-logical-pathname-translations is called.</p>	<p>Will also look in files returned by <b>logical-pathname-translations-database-pathnames</b>. See the description of that function for details.</p>