

Allegro CL 8.0 Documentation Introduction and Overview

This document contains the following sections:

[1.0 Documentation introduction](#)

[1.1 Common Graphics and Integrated Development Environment Documentation](#)

[1.2 ANSI Common Lisp and MetaObject Protocol Documentation](#)

[1.3 Some notation](#)

[2.0 The Overview section](#)

[2.1 Allegro CL documentation table of contents](#)

[2.2 Allegro CL documentation: list of overview HTML documents](#)

[3.0 The Reference section](#)

[4.0 Assistance available on the internet and the World Wide Web](#)

[4.1 The Allegro CL FAQ](#)

[4.2 Patches](#)

[4.3 Further patch information](#)

[4.4 Should you get all patches?](#)

[4.5 What to do with patches when you have downloaded them](#)

[4.6 Updating the bundle file](#)

[5.0 Reporting bugs](#)

[5.1 Where to report bugs and send questions](#)

[6.0 Contacting Franz Inc. for sales, licensing assistance, and information](#)

Documentation revision

The 8.0 documentation indicates, on each page, whether the page was revised compared to the Allegro CL 7.0 Updated Documentation. The extent of the revision is indicated in the upper right corner where it says **Allegro CL version 8.0**, also says one of:

- **Unrevised from 7.0 to 8.0:** the contents of the page have not been modified.
- **Minimally revised from 7.0:** typos and other obvious mistakes have been corrected.
- **Moderately revised from 7.0:** the revisions add new information or correct previously incorrect information.
- **Significantly revised from 7.0:** the contents include major additions or have been more or less wholly rewritten. There is a note on each significantly revised page saying what has been added or changed.

- **This page is new in 8.0:** the page was not in the Allegro CL 7.0 online documentation.
- **Page is generated automatically from other data and has been regenerated in 8.0.** The page is generated anew with each release from other data (examples include the index pages and the permuted index pages) and so is necessarily different from its counterpart in the previous release.
- **Object described on page has changed in 8.0:** the object being described has been modified in the 8.0 release and the page has changed to reflect those modifications (and perhaps in other ways). The change is always described in *release-notes.htm* (search for the object name in that document). Typical changes are new or changed arguments or new argument defaults (for operators), or a different initial value (for variables), but can be anything, including removal of the object from the system.

Allegro CL documentation is regularly updated between releases. [This page](#) on the Franz Inc. website (<http://www.franz.com/>) has links to the latest version of the Allegro CL documentation and says whether there has been an update, and, if so, when. Links on that page direct you to instructions for downloading the revised documentation if an update is available.

Beta Notice

This is the 8.0.beta documentation. It has not been fully updated.

- The indexes are unchanged and therefore somewhat out of date.
- The Common Graphics and IDE essay documentation (see *cgide.htm*) is essentially unchanged as is the IDE User Guide (see *Ide User Guide*) but there have been few changes in the interface so they are for the most part correct.
- Pictures have generally not been updated even when the text has been changed.
- There remain labels identifying a document as a 7.0 document when that should say 8.0.

Index

The main page for the **index** for the Allegro CL documentation is *index.htm*. Every documentation HTML page has a link to the index on the Navigation Bar at the top and bottom. *index.htm* points to index pages for each letter (some letters are combined). There are pages with Common Graphics functionality and pages without Common Graphics functionality.

Permuted Index

The main page for the **permuted-index** for the Allegro CL documentation is *permuted-index.htm*. Every documentation HTML page has a link to the permuted index on the Navigation Bar at the top and bottom. *permuted-index.htm* points to index pages for each letter (some letters are combined).

The permuted index is generated from the link text of the index pages. Every word in such link text is

indexed. For example, the link to the function **build-lisp-image** appears under **B** (for **build**), **L** (for **lisp**) and **I** (for **image**). This makes the permuted index a very useful tool for finding information on any topic where you know keywords but do not know which specifically appears in the index or the first letter of the names of the relevant symbols.

The permuted index has entries for each section heading of the essay documentation (those in the same directory as this file). Since they are indexed by each word in the heading, finding a section of interest using a keyword in the heading is relatively easy with the permuted index. (The headings are not in the index because they mostly start with "a" or "about" or "the" which are not words actually associated with the topic covered.)

Table of Contents

The **table of contents** of most of the essay documents is in *contents.htm*. Every documentation HTML page has a link to *contents.htm* (labeled **ToC**) on the Navigation Bars at the top and bottom. The following documents are not included in *contents.htm*: *aserve/aserve.html*, *debugger-api.htm*, *phtml.htm*, *pxml.htm*, *using-webactions.html*, and *webactions.html*.

1.0 Documentation introduction

The documentation for Allegro CL 8.0 is online. PDF versions of the HTML files in the *doc/* directory and in the *cg/* subdirectory (but not for files in the various subdirectories containing documentation pages, like *operators/*, *classes/*, *variables/*, etc.) are available in the *doc/* subdirectory of the distribution CD.

The online documentation is arranged into an [overview section](#) and a [reference section](#).

Installation information can be found in *installation.htm*.

The release notes for 8.0 can be found in *release-notes.htm*.

The User Guide for the Emacs-Lisp interface is in *eli.htm*.

Every documentation page has a navigation bar at the top and bottom with links to the table of contents, *contents.htm*; the documentation overview (this file); *cgide.htm*, which contains, in its first section, the list of Common Graphics and IDE documentation; the Release Notes, *release-notes.htm*; and the index, *index.htm*. When the file referred to is open, the link is disabled and the background color is yellow instead of aqua (you may see different colors, of course).

1.1 Common Graphics and Integrated Development Environment Documentation

The documentation for **Common Graphics** and the **Integrated Development Environment** is integrated with the general Allegro CL documentation. Start with *cgide.htm*. The section *About Common Graphics and IDE documentation* in that document provides a map of CG and IDE documentation.

1.2 ANSI Common Lisp and MetaObject Protocol Documentation

The ANSI specification is reproduced in **ANSI Common Lisp** ([ansi/ansicl.htm](#)). See also *Allegro CL and the ANSI CL standard* and *Compliance with the ANSI specification in implementation.htm*.

Documentation for the MetaObject Protocol (MOP) is in *mop/contents.html* (the link is to the initial page). The files in the *mop/* subdirectory are an HTML version of the description of the MetaObject Protocol in chapters 5 and 6 of *The Art of MetaObject Protocol* by G. Koczales, J. des Rivieres, and D. Brobrow. The HTML version was prepared by Howard Stearns and copyrighted by the Elwood Corporation, as described in *mop/admin.html*.

1.3 Some notation

Allegro directory	Used two ways: (1) the directory in which Allegro CL is installed, and (2) the translation of the <code>sys:</code> logical host, which is the directory where files needed while Allegro CL is running are located. We use the same term for both because these are normally the same directory. (<code>sys:</code> in fact translates to the directory containing the Allegro CL executable, usually <i>mlisp</i> or <i>mlisp.exe</i> , usually located in the directory into which Allegro CL was installed.)
-------------------	---

Allegro executable or executable	The file which is invoked to run Allegro CL. This is a small file and is not usually created by users (but see <i>main.htm</i>). The executable can be copied and/or renamed as desired, however. Starting with Allegro CL 6.0, there are a number of executables provided, each with an associated image file: <i>mlisp</i> (modern, case-sensitive lower, supports international character sets), <i>mlisp8</i> (modern, case-sensitive lower, does not support international character sets), <i>alisp</i> (ansi, case-insensitive upper, supports international character sets), <i>alisp8</i> (ansi, case-insensitive upper, does not support international character sets). On Windows, the executables have type <i>.exe</i> .
Allegro image file or image	This file contains the bulk of Lisp data. It is typically large and has the extension <i>dxi</i> . The executable can only be run with an image file. Image files can be created by users (with, e.g., build-lisp-image and dumplisp).

2.0 The Overview section

Html files in the *doc* subdirectory of the Allegro directory (the directory where Allegro CL and related products were installed) describe the various features of Allegro CL. There are links where necessary between these files and relevant files in the reference section.

PDF versions of these files will be available in the *doc/* subdirectory of the CD.

2.1 Allegro CL documentation table of contents

The document *contents.htm* is an outline of the Allegro CL documentation in table of contents format. The outline is too large to reasonably be included in this document.

2.2 Allegro CL documentation: list of overview HTML documents

Here is a list of the HTML (and several PDF) files that make up the base Allegro CL documentation.

To view the PDF files (such as *clim-ug.pdf*), you must have an Adobe Acrobat (r) Reader. This is

available free from Adobe. Go to this location for further information: www.adobe.com/products/acrobat/readstep.html.

<i>aodbc.htm</i>	This file describes version 2 of Allegro ODBC.
<i>aserve/aserve.html</i>	AllegroServe is a Web Server facility. See also <i>aserve/tutorial.html</i> and <i>aserve/htmlgen.html</i> . These documents are not listed in <i>contents.htm</i> nor in <i>index.htm</i> .
<i>basic-lisp-techniques.pdf</i>	<p>This document (in pdf format) is an introduction to Lisp and Lisp programming written by David Cooper.</p> <p>To view the PDF files (such as this one), you must have an Adobe Acrobat (r) Reader. This is available free from Adobe. Go to this location for further information: www.adobe.com/products/acrobat/readstep.html.</p>
<i>building-images.htm</i>	This document describes the build-lisp-image functionality and describes how to make custom images (i.e. dxl files) configured for your purposes. The function build-lisp-image , described in this document, creates a new image from scratch, inheriting a few features but no functionality from the running image (the one in which build-lisp-image is called). Note that creating an image which contains the functionality of the running Lisp image is done with dumplisp (see dumplisp) while creating images for delivery is done with generate-application , described in <i>delivery.htm</i> .
<i>case.htm</i>	As an extension to standard Common Lisp, Allegro CL permits case sensitivity. This document discusses the issues with case sensitivity.
<i>cgtk-relnotes.html</i>	[Linux only] Information on the GTK version of Common Graphics and the Integrated Development Environment (IDE).
<i>cgide.htm</i>	[Windows only] This is the introductory document to the Common Graphics and the Integrated Development Environment (IDE) documentation. It contains a description of the documentation and the IDE interface (with links), and some essays on aspects of Common Graphics and the Integrated Development Environment.

<i>clim-ug.pdf</i>	<p>This document is the CLIM User Guide, which was provided as a printed (hardcopy) document in earlier releases and, starting with release 6.0, is a PDF file. Assuming you have an Adobe Acrobat Reader, clicking on the link will display the CLIM User Guide in your browser. Note that the table of contents and the Index contain active links. Note that CLIM is an add-on product and you may not be licensed to use it.</p> <p>To view the PDF files (such as this one), you must have an Adobe Acrobat (r) Reader. This is available free from Adobe. Go to this location for further information: www.adobe.com/products/acrobat/readstep.html.</p>
<i>compiler-explanations.htm</i>	<p>The compiler has an <code>:explain</code> feature that causes it to explain the actions it takes (with regard to boxing and inlining, etc.) This document describes the explanations provided by the compiler. It also discusses inlining.</p>
<i>compiling.htm</i>	<p>This document describes the compiler. It provides information on how code generated by the compiler is controlled by the safety, space, speed, and debug optimization qualities. Information is also provided on declarations that will speed up code and tools for determining the effectiveness of declarations.</p>
<i>composer.htm</i>	<p>This document describes the Allegro Composer utility (available on UNIX platforms only). Allegro Composer provides windowized tools for development.</p>
<i>cross-reference.htm</i>	<p>This document describes the cross reference facility in Allegro CL. This facility can analyze Lisp code and determine what functions call what other functions and what functions are called by other functions.</p>
<i>debugger-api.htm</i>	<p>This document provides information about internals of the Allegro CL debugger to assist application writers who wish to provide customized debugging facilities in their applications. Note that this document does not contain information needed for ordinary use of Allegro CL. This document is not listed in <i>contents.htm</i>.</p>
<i>debugging.htm</i>	<p>This document describes the debugger in Allegro CL. The commands that provide debugging information and features of the debugger are discussed.</p>

<i>defsystem.htm</i>	Defsystem is a facility for managing files associated with an application. (It allows specifying the order in which files should be processed and the dependence of one file on another etc.) This document describes the defsystem facility in Allegro CL.
<i>delivery.htm</i>	This document describes how to deliver an application written in Allegro CL. There is also a description of how to establish a patch system for your application which complements the patch system for Allegro CL, and other delivery-related issues.
<i>dll.htm</i>	This document points to and describes examples showing how to create Lisp-based DLL on Windows. A Lisp-based DLL can be used by developers writing C, C++, and Java applications. (<i>unix-shared-library.htm</i> is a similar document describing creating Lisp-based shared libraries on UNIX platforms.)
<i>dns.htm</i>	This document describes Allegro CL support for for Domain Naming Systems.
<i>dom.htm</i>	This document describes Document Object Module (DOM) support in Allegro CL.
<i>dumplisp.htm</i>	dumplisp creates a new Lisp image file. However, the file created by dumplisp contains most of the functionality present in the currently running images, so all defined functionality will still be defined, and the current values of variables will be retained -- for the most part, see the document for details. (Contrast this with build-lisp-image which produces a fresh image which inherits very little from the running Lisp image.) This document describes dumplisp functionality and discusses issues of importance when using dumplisp .
<i>eli.htm</i>	This document describes the Emacs-Lisp interface provided with Allegro CL. See also <i>Running Lisp as a subprocess of Emacs</i> for information on connecting Emacs to Allegro CL.
<i>environments.htm</i>	This document describes the environments functionality. Based on the environments proposal described in section 8.5 of <i>Common Lisp: the Language</i> 2nd ed. (but not included in the standard eventually adopted), the environments functionality allows programmers to better describe the environments to the compiler.

errata	For errata in the documentation and corrections, see http://www.franz.com/support/8.0/index.lhtml .
<i>errors.htm</i>	Some common errors which may be encountered while using Allegro CL are discussed, along with, where possible, suggestions for fixing them. The condition system type hierarchy in Allegro CL is also discussed.
<i>examples.htm</i>	This document has some general information on Allegro CL examples, provides some links to sources of examples, and lists and (briefly) discusses the examples in the <i>examples/</i> subdirectory of the Allegro directory.
<i>flavors.htm</i>	Flavors is an object-oriented system for use in Lisp. It predates and has been effectively replaced by CLOS. We maintain flavors for backward-compatibility only. The Allegro CL flavors implementation is described in this document. Note that the reference portion (definitions of functions, macros, variables, etc.) is included with the text of this document. (Most functionality is described in the reference section of the documentation rather than being integrated into these html documents.) This document is unchanged for in this release.
<i>foreign-functions.htm</i>	The foreign function interface allows compiled foreign code to be loaded into a Lisp image and run from Lisp. (<i>Foreign</i> means C, C++, and Fortran, along with other C compatible object files.) Because of differing internal representations of data, it is not entirely straightforward to pass data from Lisp to C, to call foreign functions from Lisp, and to call back to Lisp from a foreign function. This document describes the tools provided for defining and calling foreign code.
<i>ftp.htm</i>	This document describes the Allegro FTP Client module. It can be used to communicate with an FTP server.
<i>ftype.htm</i>	This document describes how foreign types (e.g. C longs and shorts, C strings, C structures, etc.) can be defined in Lisp and how Lisp can access and operate on instances of foreign types.
<i>fwrappers-and-advice.htm</i>	The new fwrapper facility and the old, deprecated advice facility, both allow adding code that is run around functions (in some ways similar to around methods). This document describes how to use these facilities and has links to the relevant reference documentation.

<i>gc.htm</i>	Lisp maintains a garbage collector which regularly frees up space in the Lisp heap which is no longer used. This process is called garbage collection. While garbage collection is fully automatic and so a user need not necessarily think about it, configuring the garbage collector to a specific application or to a pattern of Lisp use often results in significantly better performance.
<i>gray-streams.htm</i>	This file provides details of the implementation of Gray streams in Allegro CL. Gray streams are the older stream implementation. Starting with release 6.0 simple-streams, which have advantages over Gray streams, is the primary stream implementation, but Gray streams are still supported. Simple-streams are described in <i>streams.htm</i> .
<i>../gtk/readme.txt</i>	GTK is a graphics toolkit for X windows on various UNIX machines (and on Windows). Allegro CL 8.0 has separate interfaces to GTK+ 1.2 and GTK+ 2.0, available so far on Linux and Solaris machines. The referenced <i>readme.txt</i> has further information.
<i>iacl.htm</i>	This document describes International character set support in Allegro CL. Allegro CL can support international character sets on all platforms. Note that non-international executables are also provided -- <i>mlisp8</i> for example.
<i>imap.htm</i>	This document describes the support for a client-server protocol for processing electronic mail boxes.
<i>implementation.htm</i>	This document provides specific information about the implementation of Allegro CL. Many details of the Common Lisp standard are implementation-dependent (such as the number of distinct floating-point types and their mapping to machine floats, the largest fixnum, the implementation of random, etc.) and this document says what Allegro CL does. Also discussed are extensions to Common Lisp functions (such as open, make-array, and others where an Allegro-specific enhancement is provided).
<i>index.htm</i>	This document provides links to other documents, including links to all Allegro CL functions, variables, etc.
<i>inspector.htm</i>	This document describes the non-windowized inspector. (Users on Windows with the Integrated Development Environment get all the functionality described in this document in inspector windows.)

<i>installation.htm</i>	This document describes how to install Allegro CL and related products.
<i>introduction.htm</i>	The document you are now reading.
<i>jlinker.htm</i>	This document describes a Java/Lisp interface available with Allegro CL.
<i>jil.htm</i>	This document describes a Java in Lisp facility in Allegro CL. Java in Lisp (jil) is a language for writing programs to run on the Java Virtual Machine (jvm). It uses a syntax familiar to Lisp programmers.
<i>loading.htm</i>	This document describes details of the implementation of <code>cl:load</code> (such as where is a file with no path information looked for). <code>cl:require</code> is also discussed.
<i>main.htm</i>	Allegro CL allows you to define your own C <code>main()</code> , as described in this document.
<i>miscellaneous.htm</i>	This document describes functionality too limited to need a separate document.
<i>mop/contents.html</i>	<p>The files in the <i>mop/</i> subdirectory are an HTML version of the description of the MetaObject Protocol in chapters 5 and 6 of <i>The Art of MetaObject Protocol</i> by G. Koczales, J. des Rivieres, and D. Brobrow. The HTML version was prepared by Howard Stearns and copyrighted by the Elwood Corporation, as described in <i>mop/admin.html</i>.</p> <p>We thank Mr. Stearns and the Elwood Corporation for their kind permission in allowing Franz Inc. to use these files in our product. Much useful information about Lisp, including these files, can be found on the Association of Lisp Users (ALU) home page, maintained by the Elwood Corporation, at http://www.elwoodcorp.com/alu/.</p>
<i>multiprocessing.htm</i>	Allegro CL supports multiprocessing using lightweight processes (within the running Lisp process) on UNIX and multiprocessing using OS threads on Windows. The functionality is described in this document.
<i>ole.htm</i>	A document describing interfacing to OLE. Allegro CL for Windows only.

<i>mysql.htm</i>	This document describes the new facility that allows Lisp to connect directly to MySQL. MySQL is a powerful, efficient, production ready open-source database.
<i>../opengl/readme.txt</i>	OpenGL is an open graphics library. An interface to OpenGL from Allegro CL was generated by SWIG. Layered upon this interface are a GTK and Common Graphics (Windows-only) veneer. The readme.txt file referenced has a brief introduction to the interface. More information is in these files: on the Common Graphics veneer: <i>../opengl/cggl/doc.txt</i> (Windows only). More information is in these files: on the GTK veneer: <i>../opengl/gtkgl/doc.txt</i> . (We also have an interface to GTK, mentioned above -- see <i>../gtk/readme.txt</i> . The GTK veneer on OpenGL provides OpenGL additions in GTK style.)
<i>oracle-interface.htm</i>	This document describes the new Allegro Oracle Direct Connect interface to Oracle databases.
orblink	Allegro CL includes an Orblink implementation. Its is documented in its own directory. Start with orblink/readme.htm in the <i>orblink</i> directory.
<i>os-interface.htm</i>	This document describes how to run OS functionality from within Lisp and also how to find out the current directory, how to change the current directory, and other aspects of interfacing with the operating system.
<i>packages.htm</i>	This document provides details of the implementation of packages in Allegro CL and includes a description of package locking (which prevents accidental redefinition of Common Lisp and Allegro CL functionality).
<i>pathnames.htm</i>	This document provides information on the implementation of pathnames (including logical pathnames) in Allegro CL.
plugin.htm [removed document]	This document in earlier releases described example code for implementing a Common Graphics application as a plugin for HTML browser programs such as Internet Explorer and Netscape Navigator. The example has been removed. We have encountered difficulty in making this functionality robust, and have decided not to attempt to support it in the current release.

<i>prolog.html</i>	This document describes the implementation of Allegro Prolog in Allegro CL. The documentation is preliminary. It assumes prior knowledge of the Prolog language.
<i>phtml.htm</i> and <i>pxml.htm</i>	These files (which are quite preliminary) describe the Lisp-based HTML and XML parsers. Updates to these files will be provided over time. These documents are not listed in <i>contents.htm</i> .
<i>regexp.htm</i>	This document describes the regular expression APIs available in Allegro CL. There is an older and a newer interface and both are described in this document.
<i>release-notes.htm</i>	This document is the release notes for Allegro CL on all platforms. Please refer to this document if you see unexpected behavior.
<i>rpc.htm</i>	This document describes the Remote Procedure Call (rpc) utility.
<i>runtime.htm</i>	Runtime is the technical solution to the restriction of runtime licenses. It has both legal and programming meanings. The legal meaning (determining, e.g., to whom a runtime image can be distributed and at what royalty) is defined in the Allegro CL Runtime License Agreement. In this document, the programming meaning is discussed. Information is provided on how to create runtime images (but also see <i>delivery.htm</i> and <i>building-images.htm</i>), what are the limitations of a runtime image, and so on.
<i>runtime-analyzer.htm</i>	The runtime analyzer is a tool for determining the time and space usage of code. With this tool, programs can be made more efficient and bottlenecks can be identified. This document describes the runtime analysis tools, which are statistical samplers of time and space usage and a function call counting profiler. Note that not all versions of Allegro CL include the runtime analyzer. (In earlier releases, this tool was called the profiler.)
<i>sax.htm</i>	The sax module provides a validating parser for XML 1.0 and XML 1.1. The interface to the parser based on the SAX (Simple API for XML) specification. Users provide methods for the various generic functions that implement the parser.
<i>shell-module.htm</i>	The shell module is intended to provide UNIX shell-like commands, such as you find on a modern UNIX system, and in addition shortcuts for some common Perl idioms.

<i>socket.htm</i>	Sockets are a way for different running programs to communicate with each other. This document describes the support for sockets within Allegro CL.
<i>soap.htm</i>	This document describes the Allegro CL/SOAP API.
<i>source-file-recording.htm</i>	Allegro CL provides a facility for remembering the file in which a Lisp function (or variable, parameter, macro, etc.) is defined. That facility is described in this document.
<i>startup.htm</i>	This document describes starting Allegro CL. Various methods of starting (from a shell, as a subprocess of emacs, on Windows with or without the Integrated Development Environment, etc.) are discussed. Command-line arguments and initializations files (both how to use them and how to suppress them) are described along with other things you should know about startup. Startup problems and suggested solutions are also discussed.
<i>streams.htm</i>	This file provides details of the implementation of simple-streams in Allegro CL. Simple-streams are the new standard stream type. Earlier releases of Allegro CL used Gray streams, which are still supported. See <i>gray-streams.htm</i> .
<i>test-harness.htm</i>	This document describes the test harness facility. A test harness is a collection of macros and variables associated with testing programs and applications, along with templates for test forms.
<i>top-level.htm</i>	The top level is the interface to a running development Lisp image. (We say `development' to distinguish from a Lisp application. Most applications provide their own top level.) This document provides information about the Allegro CL top level.
<i>unix-shared-library.htm</i>	This document points to and describes examples showing how to create Lisp-based shared libraries on UNIX platforms. A Lisp-based shared library can be used by developers writing C, C++, and Java applications. (<i>dll.htm</i> is a similar document describing creating a Lisp-based Windows DLL.)
<i>uri.htm</i>	This document describes the URI facility. URIs are a superset in functionality and syntax to URLs (Universal Resource Locators) and URNs (Universal Resource Names). Allegro CL provides support for manipulating URIs.

<i>version-70-release-notes.htm</i>	This document is the release notes supplied with release 7.0. Most of the information it contains is not repeated in <i>release-notes.htm</i> . Note that some of the information does not apply to release 8.0.
<i>webactions.html</i>	This is the user guide and reference manual for the Allegro Webactions facility for generating dynamic web pages. See also <i>using-webactions.html</i> , which is a general introduction. Webactions works with AllegroServe (see <i>aserve/aserve.html</i>).
<i>xml-rpc.htm</i>	This document describes the Allegro CL implementation of XML-RPC, which implements classes and functions to support the XML-RPC protocol for remote function calls through HTTP connections.

3.0 The Reference section

For Allegro CL-specific functionality, virtually every operator (function, generic function, or macro), every constant and variable, and many classes defined in Allegro CL, other than standard Common Lisp functionality, are described in individual HTML files called *description pages*. These files are arranged as follows:

[Allegro directory]/doc/[kind]/[package]/[symbol].htm

Where:

[Allegro directory] is the directory into which Allegro CL was installed. The document you are reading is *[Allegro directory]/doc/introduction.htm*.

[kind] is one of

- operators (for functions, generic functions, and macros)
- variables (for variables and constants)
- classes
- other (currently, top-level commands, ide-menus-and-dialogs)

[package] is the home package of the symbol being documented, and is one of

common-graphics, compiler, composer, dbi, defsys, excl, ff, javatools.jil, javatools.jlinker, mp, net.post-office, net.rpc, net.uri, prof, socket, system,

top-level, util.test, xref

Note that some packages have no entries under certain kinds.

[symbol] is the symbol naming the object (or the name of the top-level command). A * character in a symbol name is handled specially. * appears as s_ or _s depending on whether the * is leading or trailing.

Thus, the documentation for the variable **excl:*enable-package-locked-errors*** is in the file:

[Allegro directory]/doc/variables/excl/s_enable-package-locked-errors_s.htm

And the documentation for the function **system:command-line-arguments** is in the file

[Allegro directory]/doc/operators/system/command-line-arguments.htm

Note: Common-graphics operators are further organized by the first letter of the symbol naming them. This prevents a directory with over 1600 files.

Each page names the symbol, its home package, and the type of the object being described (function, macro, variable, class, etc.)

Descriptions of operators include the argument list. Generally the argument list is the same as returned by functions like **arglist** but not always. It will not be when (1) the stored argument list is abbreviated (e.g. &rest args or &key &allow-other-keys, where argument processing is done within the operator body) and thus unhelpful; or (2) the stored argument names are unhelpful (*struct* -- because the object is a struct -- rather than, say, *process* or *pathname*).

Some arguments are listed but described as not for programmer use (these arguments may be intended for later enhancements or may support internal actions only). A few pages describe internal functionality not intended for programmer use. Symbols naming such functionality are exported for unavoidable system reasons and because they are exported, they have description pages.

4.0 Assistance available on the internet and the World Wide Web

Franz Inc. maintains a World Wide Web home page (<http://www.franz.com/>). Along with much else, the latest update of the Allegro CL documentation, the latest Allegro CL FAQ, and the latest Allegro CL patches can be found on the Franz Inc. WWW home page.

4.1 The Allegro CL FAQ

FAQ stands for *Frequently Asked Questions*. The **Allegro CL FAQ** is a document written in question and answer format containing answers to questions often asked by users of Allegro CL and related products. Franz Inc. updates the FAQ regularly.

The FAQ is available on the Franz Inc. WWW home page (in the **Support** section):

<http://www.franz.com/>

4.2 Patches

A patch is a file (typically a *fasl* -- compiled Lisp -- file) which either corrects some error in the product or provides some enhancement or new feature. Patches are available from the Franz Inc. web site. (Some patches, particularly those which implement an improvement, an enhancement, or a new feature rather than a bug fix, are restricted to supported customers only.) Patches are stored in both locations with a directory structure that mirrors the distribution directory structure, so that patch files can be downloaded into the correct directories of the distribution. The function **update-allegro** downloads patch files and places them in the correct locations.

There are various kinds of patch files.

- Files with names like *p8a002.001*. These are called *patches* and are compiled Lisp files (despite not having the extension *fasl*) which fixes some specific problem or provides an enhancement or, sometimes, adds a new feature). The names of these files encode information about their applicability and use. Please do not rename them. The extension indicates the patch version. Patches are often replaced (perhaps to provide additional fixes, perhaps to fix an introduced problem). When that happens, a new version (with a different extension). The patch system ensures you have a consistent set of patches. These patch files go in the *update/* directory of the Allegro directory. Note the second character ("8" in the example) identifies the version of Allegro CL and is the value of `excl::*cl-patch-version-char*`.)
- Files with module names and three-digit numeric extensions (top-level.001 or streams.003). These files replace Lisp library files in the *code/* subdirectory of the Allegro directory. (Most of these files are originally in the Lisp bundle file and so you may not see the originals of these files.) The system knows to look in the *code/* subdirectory before looking in the bundle file. The names of these files must also not be changed.

- Miscellaneous documentation and other files. Documentation files may be updated from time to time and **update-allegro** will download these files as well.

On the web site, <http://www.franz.com/>, choose Support, then click on the link to patches and follow the instructions.

4.3 Further patch information

Note the following points.

- The patch files are numbered and coded by product, and they are read in order, higher numbers first. The order in which they are read is very important. Do not change the names of patch *fasl* files. A single patch may have various versions. This means the patch has been updated.
 - Do not use a patch for one release of Allegro CL with another release. (Patches are placed in a subdirectory of the Allegro directory. Since each new release (typically) has a different Allegro directory, you can have more than one distribution on a single machine, and patches in each one. If you do that, be sure that patches for a particular release are placed only in that release's Allegro directory.)
-

4.4 Should you get all patches?

Whether or not you should get all available patches depends on your current tolerance for instability. Because patches are less well tested than releases, patches may occasionally introduce errors as well as fix problems.

However, even a correct patch may introduce instability: if a patch enables a feature which was previously ignored or signals an error (correctly) where none was previously signaled, your code may fail because the patch uncovered a problem which was previously unnoticed. Suppose, for example, a patch causes certain declarations to be used during compilation (without the patch they are ignored). Such a patch would not fix a bug (since ignoring declarations is permitted) but if your code happened to contain incorrect declarations, then the patch would be destabilizing. (Without the patch, the wrong declarations were ignored and so did not harm. With the patch, they are used.)

The potential impact of a patch is given in the LOG file in the patch directory. Here are two entries from the 5.0 patch LOG (the 8.0 patch LOG file will be organized similarly):

```
Mon Sep 14 11:27:52 PDT 1998
```

Patch: update/p0a001.001

Fixes a bug where find-restarts, when given the optional condition argument, would not consider restarts that are not associated with any condition.

Impact: Should be minor

Tue Sep 1 14:33:29 PDT 1998

Patch: update/p0b002.001

Fixes self tail recursive call with intermixed args.

Problem resulted in errors in compiled code, not in interpreted.

Impact: Recommended.

Notice the Impact line (made bold for emphasis). It provides an assessment of how destabilizing a patch may be. Note that in these cases, one is minor and the other patch (which fixes a problem where compiled code ran incorrectly) is recommended.

If you are in a development cycle, our advice is to get all available patches for a platform (machine type) and Allegro CL version. Patches for associated products (such as CLIM) should be included as well.

But if you are preparing a delivery, we advise you to be selective, perhaps getting only the patches that deal with problems you report.

Note that we have previously advised even users who are developing applications rather than preparing for delivery to only include patches when they experienced the problem fixed by the patch. There are pluses and minuses to each recommendation. While including all patches gives you all available fixes, as we said above, patches are not as well tested as releases and sometimes a patch introduces a new problem or bug. However, we have changed our advice for when you are developing because many patches are for performance, and therefore generally useful, and because problems introduced by patches, while they do happen, are uncommon and are usually fixed quite quickly (typically with an updated patch), and because it is relatively easy to back out a patch if it causes problems.

Note that the report created by **dribble-bug** lists all patches included in the image. It is very important that this information be included with a bug report.

4.5 What to do with patches when you have downloaded them

update-allegro downloads patches but does not create new images. To create new images, exit Lisp (if you are running it) and run *update.sh* on UNIX machines and *update.bat* on Windows machines. These scripts will update all Allegro-supplied image (dxl) files by starting Lisp with the dxl file and calling

build-lisp-image. They also make backups of the the dxl files. You must recreate any image files you created yourself.

Note on backups of dxl files: the first time *update.sh/bat* is run, the dxl files are copied to files with ``orig'` added -- *mlisp.dxl* to *mlisporig.dxl*, e.g. The second time, they are copied with ``old'` added -- *mlisp.dxl* to *mlispold.dxl*, e.g. On subsequent runs, the backup may not be done correctly. Please move **old.dxl* to **oldn.dxl* (for some *n* -- for example, *mlispold.dxl* to *mlispold1.dxl*) before updating to ensure that the backups are done correctly.

Backing out of patches: you may occasionally discover that a patched image does not work properly. (Perhaps the patch was faulty, or perhaps the fix, whatever it is, interacts incorrectly with your application.) To back out of patches, (1) rename the new dxl files, (2) copy the original dxl files (they will have "old" or "orig" appended to their filenames) to their original names, and (3) move any *new* fasl files out of the *code/* subdirectory. You are now back to where you were before running *update.sh* or *update.bat*.

4.6 Updating the bundle file

The bundle file contains a collection of fasl files associated with Allegro CL modules. When a module is autoloaded or loaded by a call to **require**, it is typically loaded from the bundle file. (If you see the message "Fast loading from bundle...", the file is being loaded from the bundle file. The location and name of the bundle file are returned by the function **bundle-pathname**.

Sometimes module fasl files have to be updated. In that case, a new file is downloaded from the Franz Inc. patch repository and is placed in the *code/* subdirectory of the Allegro directory. The system always checks the code directory first for a module fasl file before looking in the bundle file. That assures that the latest version of a module fasl file is loaded.

However, it is often useful to update the bundle file to include all the updated module fasl files. In Allegro CL 8.0, this update is performed automatically (when **update.sh** or **update.bat** is run).

5.0 Reporting bugs

Before reporting a bug, please study this document and the ANSI CL Standard document.

A report that such and such happened is generally of limited value in determining the cause of a

problem. It is very important for us to know what happened before the error occurred: what you typed in, what Allegro CL typed out. A verbatim log may be needed. If you are able to localize the bug and reliably duplicate it with a minimal amount of code, it will greatly expedite repairs.

It is much easier to find a bug that is generated when a single isolated function is applied than a bug that is generated somewhere when an enormous application is loaded. Although we are intimately familiar with Allegro CL, you are familiar with your application and the context in which the bug was observed. Context is also important in determining whether the bug is really in Allegro CL or in something that it depends on, such as the operating system.

To this end, we request that your reports to us of bugs or of suspected bugs include the following information. If any of the information is missing, it is likely to delay or complicate our response.

- **dribble-bug or print-system-state output.** The function **dribble-bug** is like **common-lisp:dribble** but prefaces the transcript with information about the image being run (platform, OS, Allegro CL version, patches loaded, and much else). Please include the output of **dribble-bug** in the first report of any bug or problem. The function **print-system-state** prints the information about the platform, patches, etc. that goes into a dribble-bug file. You can use that function to print the data to send to us instead of using **dribble-bug**, but that information, however, obtained should be included in each bug report.
- **Information about you.** Tell us who you are, where you are and how you can be reached (an electronic mail address if you are reachable via the Internet, a postal address otherwise, and your telephone number), and in whose name the license is held. It is best to put the email address in the text of the message (in case the mailer program messes up the return address). Please include a telephone number in case an email response bounces.
- **A description of the bug or problem.** Describe clearly and concisely the behavior that you observe and why it is not what you expect or want.
- **Exhibits.** Provide us with the smallest, self-contained Lisp source fragment that will duplicate the problem, and a log (e.g. produced with **dribble-bug**) of a complete session with Allegro CL that illustrates the bug. Note that dribble output does not log certain messages printed by the Operating System or printed by foreign code called by Lisp or routines called by **run-shell-command** and related functions. If such output is relevant, please capture it in an Emacs buffer or in some similar fashion and include the output with the information you submit.

Use **dribble-bug** as follows. Entering

```
(excl:dribble-bug filename)
```

causes implementation and version information to be written to the file specified by *filename*, and then records the Lisp session in the same file. Exiting Lisp or entering

```
(dribble)
```

will close the file after the bug has been exhibited. The following dialog provides a rudimentary template for the kernel of a bug report.

```
USER(5) (dribble-bug "dribout")
USER(6) ;; Now duplicate your bug . . .
USER(7) (dribble)
```

It may be that the bug causes failure which prevents getting a dribble transcript showing the bug. We still need the dribble-bug output showing the platform, patches, etc. Get this with

```
USER(5) (dribble-bug "dribout")
USER(6) (dribble)
```

print-system-state prints the information that goes in a dribble-bug file. You can use that function to get the information if you wish. In either case, send the contents of the file *dribout* or the output of **print-system-state**.

Send bug reports to either the electronic mail or postal address given in [Section 5.1 Where to report bugs and send questions](#). We will investigate the report and inform you of its resolution in a timely manner.

As we said before, the **dribble-bug** log does not capture things printed by the operating system or by operating system utilities. These messages may be important. Please be sure you include them in the message.

5.1 Where to report bugs and send questions

Send problem reports and technical questions of any type by email to **support@franz.com**. Our mailing address is Franz Inc., Suite 1450, 555 12th St., Oakland CA 94607 USA; and our telephone number (in the USA) is +510-452-2000 and our FAX number is +510-452-0182. (Note: our address and phone numbers are new as of August 1, 2002.)

6.0 Contacting Franz Inc. for sales, licensing assistance, and information

Sales and licensing questions should be directed to your account manager. If you are not already a

customer (and thus do not yet have an account manager), send email to **info@franz.com**, or call Franz Inc. (in Oakland, California, USA) at +1 510-452-2000, or send regular mail to Franz Inc., Suite 1450, 555 12th St., Oakland, CA 94607 USA. Please give your email address, phone number, and mailing address. The appropriate Account Manager will contact you.

If you are already a customer, you should have the name and email address of your account manager, but if they are not to hand, use the general ones given just above.

General product information is available on the Franz Inc. web site, <http://www.franz.com/>. You can get additional information by contacting us. Again, send email to **info@franz.com** or call +1 510-452-2000 or write to Franz Inc., Suite 1450, 555 12th St., Oakland, CA 94607 USA.