

Release Notes for Allegro CL 7.0

This document contains the following sections:

[1.0 Introduction](#)

[2.0 Information on changes and new stuff since the 7.0 release](#)

[3.0 All pre-7.0 Lisp compiled files must be recompiled \(old fasl files will not load\)](#)

[4.0 Release Notes for installation](#)

[5.0 Release notes for specific platforms](#)

[5.1 OS patch needed for Solaris 2.8 on Sparcs](#)

[5.2 HP Alpha running Tru64: default stack size should be increased](#)

[5.3 Notes of increasing the default maximum stack size on HP-UX 11 machines](#)

[5.4 Mac OS X notes](#)

[5.4.1 Increasing stack size on Mac OS X](#)

[5.4.2 Building shared libraries on Mac OS X](#)

[5.5 Heap start locations](#)

[6.0 Release Notes for the base Lisp](#)

[6.1 New features in the base Lisp](#)

[6.1.1 Features added to Allegro CL 6.2 after the initial release of Allegro CL 6.2](#)

[6.1.2 Major new features in Allegro CL 7.0](#)

[6.1.3 Other new features in Allegro CL 7.0](#)

[6.1.4 The new model for multiprocessing and OS threads in Allegro CL](#)

[6.2 Non-backward-compatible changes in the base Lisp](#)

[6.3 Other changes to and notes about the base Lisp](#)

[6.4 Base Lisp platform-specific information](#)

[7.0 Release Notes for CLIM](#)

[8.0 Release Notes for Common Graphics and the IDE \(Windows only\)](#)

[8.1 Non-backward-compatible changes in Common Graphics](#)

[8.2 Other changes in Common Graphics and the IDE](#)

[8.3 IDE release notes](#)

[8.3.1 The console window and tray icon in applications](#)

[8.3.2 Opening projects from releases prior to 7.0](#)

[8.4 New behavior when starting the IDE or creating a new project or form](#)

[9.0 Release Notes for AllegroStore](#)

[9.1 Non-backward-compatible changes in AllegroStore](#)

[9.2 Other changes in AllegroStore](#)

[10.0 Release notes for SOAP](#)

[11.0 Release notes for jLinker](#)

[12.0 Release notes for AllegroServe](#)

[13.0 Release notes for IMAP and SMTP, XMLutils](#)

[14.0 Release notes for The Emacs/Lisp interface](#)

[15.0 Documentation modifications in Allegro CL 7.0](#)

[16.0 Availability of CLX for Allegro CL](#)

[17.0 Release notes for Orblink](#)

[Appendix A. Tightening of ANSI Conformance in Allegro CL](#)

[Appendix A.1. Conformance fixes which might break existing code](#)

[Appendix A.2. Other conformance fixes](#)

[Appendix B. Details of Common Graphics/IDE package reorganization](#)

[Appendix C. The reorganization of the CG class structure](#)

[Appendix D. Details of separating DDE code from Common Graphics](#)

[Appendix E. Functionality/symbols removed or renamed in Common Graphics](#)

[Appendix F. Common Graphics compatibility with aclpc 3.0.2 has been removed](#)

[Appendix F.1. The aclwin and aclffi compatibility modules are no longer loaded by Common Graphics](#)

[Appendix G. The effects of the new, longer array implementation on def-foreign-call and def-foreign-type](#)

1.0 Introduction

This document provides the release Notes for release 7.0 of Allegro Common Lisp and related products. Many sections are divided into non-backward-compatible changes (that produce different behavior in release 7.0 compared to release 6.2) and changes unrelated to backward-compatibility. Note that a bug fix is *not* considered a backward-incompatible change even if it does result in changed behavior because the previous behavior was erroneous.

You may wish to look at the 6.2 Release Notes, included in this distribution as the file **version-62-release-notes.htm**

This document describes the changes, major and minor, from 6.2 to 7.0. Please look particularly at these sections:

- [Section 6.1 New features in the base Lisp](#): this section describes new features.
- [Section 6.2 Non-backward-compatible changes in the base Lisp](#): this very important section describes changes to Allegro CL which are **not** backward compatible. You may have to modify your source code in light of these changes.
- [Section 8.1 Non-backward-compatible changes in Common Graphics](#): [Windows customers only] this section describes changes to Common Graphics which are **not** backward compatible. You may have to modify your source code in light of these changes.
- [Section 9.1 Non-backward-compatible changes in AllegroStore](#): [AllegroStore customers only] this section describes changes to AllegroStore which are **not** backward compatible. You may have to modify your source code in light of these changes.
- [Section 2.0 Information on changes and new stuff since the 7.0 release](#): this section currently has no content. It is a placeholder for descriptions of material added after the initial Allegro CL 7.0 release, along with other post-7.0 changes. Content will only be added to the section in documentation updates after the release of Allegro CL 7.0.

If you notice changed or unexpected behavior with an operator, variable, class, or facility, search for its name in this document to see whether there is an entry concerning it.

2.0 Information on changes and new stuff since the 7.0 release

This section is a placeholder for material added in documentation updates after the 7.0 release.

3.0 All pre-7.0 Lisp compiled files must be recompiled (old fasl files will not load)

fasl files (compiled Lisp files) created by releases of Allegro CL prior to 7.0 (including 7.0 Beta) will not load into Allegro CL 7.0. All such files, including all 7.0 Beta files, must be recreated by compiling the associated Lisp source files. An error will be signaled if Lisp attempts to load an older, incompatible *fasl* file.

4.0 Release Notes for installation

1. **Version 7.0 uses the 6.2 installation procedure:** installation is described in **installation.htm**.
 2. **The distribution includes 8 bit and 16 bit character images** (this information is repeated from the 6.2 Release Notes). Allegro CL 7.0 has images that support 8 bit characters only, or 16 bit characters only. It is our expectation that most users will use the 16 bit images. Executables supplied with the distribution either have or do not have `8' in the name. Those with 8 in the name (mlisp8 and alisp8, e.g.) support 8 bit characters. Those without a number in the name support 16 bit characters. Image (dxl) files also come in 8 and 16 varieties. Again, 8 in the name means 8 bit character support. Character support for images and executables must match. Trying to start an executable with the wrong type of image fails.
 3. **No prebuilt Allegro Composer images in the distribution** (this information is repeated from the 6.2 Release Notes). To create an Allegro Composer image, start Allegro CL and load *buildcomposer.cl*. That will produce *composer* and *composer.dxl*, or *composer8* and *composer8.dxl*. Allegro Composer is available on Unix only.
 4. **No prebuilt Allegro CLIM images in the distribution** (this information is repeated from the 6.2 Release Notes). To create a CLIM image, start Allegro CL and load *buildclim.cl*. That will produce *clim* and *clim.dxl*, or *clim8* and *clim8.dxl*.
-

5.0 Release notes for specific platforms

Allegro CL 7.0 is known to work with the following minimal operating system versions. Allegro CL 7.0 runs on all operating system versions released (not in beta or pre-release form) as of October 15, 2004. See below for places to obtain information on operating systems released after that date.

32-bit platforms

- HP Tru64 UNIX 5.1 -- see [Section 5.2 HP Alpha running Tru64: default stack size should be increased](#)
- HP-UX 11.00 -- see [Section 5.3 Notes of increasing the default maximum stack size on HP-UX 11 machines](#)
- FreeBSD 5.2
- Linux (x86), glibc 2.2
- Linux (PPC), glibc 2.3
- Apple Mac OS X 10.3 -- see [Section 5.4 Mac OS X notes](#)
- Microsoft Windows 98/Me and 2000/XP/Server 2003
- IBM AIX 5.1
- Sun Solaris 2.8 -- see [Section 5.1 OS patch needed for Solaris 2.8 on Sparcs](#)

64-bit platforms

- HP Tru64 UNIX 5.1 -- see [Section 5.2 HP Alpha running Tru64: default stack size should be increased](#)
- HP-UX 11.00 -- see [Section 5.3 Notes of increasing the default maximum stack size on HP-UX 11 machines](#)
- Linux (AMD64), glibc 2.3
- IBM AIX 5.1
- Sun Solaris 2.8 -- see [Section 5.1 OS patch needed for Solaris 2.8 on Sparcs](#)

Franz Inc. cannot maintain the same release schedule as the many operating system providers on the many platforms we support. Usually Allegro Common Lisp and all of its component parts will work on new Operating System versions that become available after release. But sometimes Allegro CL patches or operating-system patches, or installation tweaks, will

be required before Lisp will run on an updated system.

Franz Inc. usually finds out about any such issues soon after new operating system releases appear and devises any necessary patches or compatibility procedures. We maintain current information about operating system versions on this web page: <http://www.franz.com/support/osinfo.lhtml>. We strongly advise you to check that page before updating your operating system. If we know that the new operating system is compatible, or is compatible with certain patches, you will find the information there. Similarly if it is known to be incompatible. If the new operating system is not yet listed it may be that it has not yet been tested.

5.1 OS patch needed for Solaris 2.8 on Sparcs

A problem with Solaris 2.8 where calls to **run-shell-command** can cause Lisp to hang is fixed by operating system patch 108993-18 (which supersedes the earlier patch 108827-36 for this problem). On Solaris 8, it can also be fixed by adding **/usr/lib/lwp** to the front of your `LD_LIBRARY_PATH` environment variable. No fix is necessary for Solaris 2.9. (The problem also affects Solaris 2.6, where an OS upgrade is needed and Solaris 2.7, fixed by patch 106980-21, but Allegro CL 7.0 is not supported on Solaris 2.6 or 2.7.)

The patch can be obtained from this Sun website: <http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert/49843>.

5.2 HP Alpha running Tru64: default stack size should be increased

The default stack size limit on an Alpha is 2 Megabytes, which is too low for normal stack overflow handling. Doing

```
limit stacksize unlimited
```

in a csh shell seems to allow up to 16 or 32 Mb, and users can run that command before running Allegro CL. (That command could be put into a `.cshrc` file.)

However, that only solves the problem for Allegro CL processes that are started from within that shell. You can also change the default for the machine as a whole by doing something like the following. (This procedure is provided as an example of what might work. Please check your Tru64 system documentation or contact your Compaq service representative for the precise instructions.)

The procedure described here is best performed by a system administrator or similarly trained person. The `vmunix` file created and copied at the end is the UNIX kernel. Modifying it incorrectly or corrupting it can have serious consequences.

- 1. Log in as root.** You must be root (or have superuser privileges) to perform most of the following operations.
- 2. Change into the `/sys/conf/` directory.**

```
prompt# cd /sys/conf
```

- 3. Edit the file whose name is the machine name, which we will call `MACHINE_NAME`, as follows.** (Such a file typically exists after normal Tru64 Unix install. If the file does not exist, the System Administrator will have to create one.)

Change the line

```

dflssiz          2097152
to
dflssiz          8388608
```

or

df1ssiz **16777216**

The value of **df1ssiz** may already be different than 2097152, which is 2 megabytes -- (* 2 (expt 2 20)). The new suggested values are 8388608, which is 8 megabytes and a good value for 32-bit Lisps, or 16777216, 16 megabytes and a good value for 64-bit Lisps.

4. Run `/usr/sbin/doconfig` as follows (recall that *MACHINE_NAME* is the file in */sys/conf* whose name is the machine name):

```
prompt# /usr/sbin/doconfig -c MACHINE_NAME
```

You should see output similar to:

```
*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***
```

```
Saving /sys/conf/MACHINE_NAME as /sys/conf/MACHINE_NAME.bck
```

```
Do you want to edit the configuration file? (y/n) [n]: n
```

```
*** PERFORMING KERNEL BUILD ***
```

```
Working...Thu Oct 4 09:58:16 PDT 2001
```

```
The new kernel is /sys/MACHINE_NAME/vmunix
```

5. Copy the new kernel to / with a command like the following

```
prompt# mv /sys/MACHINE_NAME/vmunix /
```

6. Reboot the system.

5.3 Notes of increasing the default maximum stack size on HP-UX 11 machines

Increasing the default maximum stack size on HP-UX 11 requires modifying kernel configuration parameters, then rebuilding the kernel. This process should be done by the systems administrator.

The file */stand/system* is the kernel configuration file. The `maxssiz` and `maxssiz_64bit` tunable parameters need to be increased to at least 32MB each. If you don't have any references to `maxssiz` or `maxssiz_64bit` in your */stand/system* file, then you can simply add these lines to the bottom of the file:

```
maxssiz          (32*1024*1024)
maxssiz_64bit    (32*1024*1024)
```

Otherwise, you'll need to modify the existing parameters so that they express a value \geq 32MB.

After modifying */stand/system*, the kernel needs to be rebuilt using the following command:

```
/usr/sbin/config -u /stand/system
```

The system will have to be rebooted for the changes to take effect.

5.4 Mac OS X notes

Allegro CL 7.0 is not supported on Mac OS X version 10.0. You must use version 10.3.

There are two other issues:

1. [Section 5.4.1 Increasing stack size on Mac OS X](#)
2. [Section 5.4.2 Building shared libraries on Mac OS X](#)

5.4.1 Increasing stack size on Mac OS X

The default stack size limit on a Mac OS X machine (apparently 512 Kbytes) is too low for normal stack overflow handling. Doing

```
limit stacksize unlimited
```

in a csh shell seems to allow up to 16 or 32 Mb, and users can run that command before running Allegro CL. (That command could be put into a *.cshrc* file.)

However, that only solves the problem for Allegro CL processes that are started from within that shell. You can also change the default for the machine as a whole by doing something like the following. (This procedure is provided as an example of what might work. Please check your Mac OS X system documentation or contact your Apple service representative for the precise instructions.)

The procedure described here is best performed by a system administrator or similarly trained person. The */etc/rc* file edited during the procedure is key to correct operation of the Operating System. Modifying it incorrectly or corrupting it can have serious consequences.

1. **Log in as root.** You must be root (or have superuser privileges) to perform most of the following operations.
2. **Get access to the command line shell on the computer.** This can be accomplished by launching the **Terminal** application.
3. **Open up the file */etc/rc* with a text editor.**
4. **Add the following lines near the top of the file** (the values shown, 8m and 10m, should be suitable for Allegro CL applications):

```
limit stacksize 8m
limit datasize 10m
```

The actual contents of the file differ on different machines and setups. Here is an example of a suitable location: just before the following lines:

```
##
# Handle arguments passed from init.
##

BootType=$1; shift;

if [ -z "${BootType}" ]; then BootType="multiuser"; fi
```

5. **Save the file */etc/rc*.**
6. **Reboot the machine.**

5.4.2 Building shared libraries on Mac OS X

Building shared libraries

Building shared libraries on Mac OS X in foreign-functions.htm describes how to create a shared library suitable for loading into Allegro CL. We have determined that the `-flat_namespace` to the `ld` used to create the shared library is necessary, as shown in the linked section.

5.5 Heap start locations

When building large new images, it is often useful to specify Lisp Heap and C-heap start locations. See the discussion of the `lisp-heap-start` and `c-heap-start` keyword arguments in **Arguments to build-lisp-image 2: defaults not inherited from the running image in building-images.htm**. Here are the initial locations (called `bases') in Allegro CL 7.0 as delivered. Values are Hexadecimal integers.

Operating System	Lisp base	C base
Tru64 32-bit	0x30000000	0x54000000
Tru64 64-bit	0x1000000000	0x2000000000
FreeBSD	0x10000000	0x64000000
HP-UX 32-bit	0x10000000	0x64000000
HP-UX 64-bit	0x8000001000000000	0x8000002000000000
Linux (x86)	0x71000000	0xa0000000
Linux (PPC)	0x40000000	0x74000000
Mac OS X	0x5008000	0x64000000
Windows	0x20000000	0x54000000
AIX 32-bit	0x30000000	0x64000000
AIX 64-bit	0x7000010000000000	0x7000020000000000
Solaris 32-bit	0x40000000	0x54000000
Solaris 64-bit	0x1000000000	0x1000000000
AMD 64-bit	0x1000000000	0x2000000000

6.0 Release Notes for the base Lisp

This main section contains three subsections (which have additional subsections): one on new features ([Section 6.1 New features in the base Lisp](#)), one on changes which are not backwards compatible and so may require code changes, ([Section 6.2 Non-backward-compatible changes in the base Lisp](#)), and one on miscellaneous changes ([Section 6.3 Other changes to and notes about the base Lisp](#)).

6.1 New features in the base Lisp

We have added a number of new capabilities to Allegro CL. Here we give links to the documentation of the new features.

The features described in [Section 6.1.1 Features added to Allegro CL 6.2 after the initial release of Allegro CL 6.2](#) were released originally as patches to Allegro CL 6.2, but after the initial release of Allegro CL 6.2. The features described in [Section 6.1.2 Major new features in Allegro CL 7.0](#) are new in the Allegro CL 7.0 release.

6.1.1 Features added to Allegro CL 6.2 after the initial release of Allegro CL 6.2

New shell module

The shell module is intended to provide UNIX shell-like commands, such as you find on a modern UNIX system, and in addition short cuts for some common Perl idioms. This module does not provide functionality to spawn a shell (as older functions like **run-shell-command** and **shell** and newer ones like **command-output** do). Instead, many commands that are available in a shell are available in this module. These commands are available on UNIX and Windows and work in a uniform fashion whatever the actual operating system. No external programs are used to implement the functions in this module, and as a result security issues associated with executing external programs by spawning shells are finessed.

Associated with the shell module is a new implementation of **system:with-command-line-arguments**. The new implementation is backward compatible, so current calls to the macro will work as before. The new implementation provides more options and the ability to use long form options.

See also **Pathname wildcard syntax** in **pathnames.htm**. This newly added section provides information on how wildcard characters (like * and ?) in pathnames are handled.

Allegro FTP Client

The Allegro FTP Client module has been improved. Various versions of this software have been previously available (as described in the documentation, **ftp.htm**). The module can be used to communicate with an FTP server. See **ftp.htm** for more information.

Allegro MySQL

MySQL is a powerful, efficient, production ready open-source database that runs on all popular platforms. There are language bindings for most popular languages. A new module, described in **mysql.htm** provides an interface from Allegro CL to MySQL.

Note that the module has undergone changes since its original release and there are additional changes in the 7.0 release. One change provides improved handling of external formats. **connect** now has an external-format keyword argument and the function **mysql-external-format** polls (and with **setf** sets) the current external format for a connection.

New OSI (Operating System Interface) Module

A new module, `:osi` provides more connections to Operating System functionality. The new module is described in **The Operating System Interface (OSI) module** in **os-interface.htm**. The goal of the OSI module is to provide a Lisp layer for operating system services available on the platforms on which ACL runs. The operators, constants, and classes of the new module are not described (in this documentation release) on individual documentation pages. Instead, the descriptions are in **Operating System Interface Functionality** in **os-interface.htm**.

As part of the changes introduced by the new OSI module, the following functions have been changed (by adding new keyword arguments, so current calls are unaffected):

- **copy-file** (new *remove-destination* and *force* keyword arguments)
- **delete-directory-and-files** (new *force* keyword argument)
- **run-shell-command** (new *directory* keyword argument).

- **cl:open** now accepts `:always-append` as the value of the *if-exists* and *if-does-not-exist*. This value means that concurrent writes by any number of programs will always write to the end of the file. This is useful for writing to log files. Be warned, however, that changing the file position of a stream opened with `:if-exists` `:always-append` or `:if-does-not-exist` `:always-append` will have no effect. See the description of the implementation of **cl:open** in **Extensions to cl:make-package, cl:disassemble, cl:open, cl:apropos** in **implementation.htm**.

Modification to **acl-socket:dns-query**

acl-socket:dns-query has been modified. There are two modifications: one causes a fourth value to be returned. That value is a list of the flags returned by the dns server that replied to the query. The second modification allows the value `:any` as a value for the *type* keyword argument (see the description of **acl-socket:dns-query** for further information). The change is essentially backward compatible (unless your code depends for some reason on exactly three values being returned by **acl-socket:dns-query**).

Also, the documentation of **acl-socket:dns-query** has been clarified to make clear when the type is `:ptr`, name can be an integer IP address or a string containing a dotted IP address (the documentation previously incorrectly said name could only be an integer IP address when type is `:ptr`).

An Allegro CL/SOAP API

Allegro CL/SOAP now supports WSDL 1.1 and WSDL generation. Please see **soap.htm** for more information on Allegro CL and SOAP.

SOAP was released in Allegro CL 6.2 in stages. The release with 7.0 includes various things not in the 6.2 releases. See [Section 10.0 Release notes for SOAP](#) for details.

An upgrade to the Emacs-Lisp Interface

There is a new version of the Emacs-Lisp interface. Among other things, the patch improves menus for Emacs 21 and later, and offers the option of complying with the major mode conventions outlined in the GNU Emacs Lisp Reference Manual, edition 2.5. See the new section **Significant changes in the interface** in **eli.htm** for more information. Some miscellaneous fixes are listed in [Section 14.0 Release notes for The Emacs/Lisp interface](#).

A validating parser for XML 1.0 and XML 1.1 in the new SAX module.

The new sax module, described in **sax.htm**, provides a validating parser for XML 1.0 and XML 1.1. The interface to the parser is based on the SAX (Simple API for XML) specification. Users provide methods for the various generic functions that implement the parser.

The asdf system definition facility now included with Allegro CL

The popular system definition facility, asdf, is now included with Allegro CL. Evaluate `(require :asdf)` to load it into a running Lisp. See [\[Allegro directory\]/code/asdf.readme](#) for more information, [\[Allegro directory\]/code/asdf.license](#) for the license, and [\[Allegro directory\]/src/asdf.lisp](#) for the source code.

Allegro Webactions

Allegro Webactions is a dynamic web page generator facility designed to be used with AllegroServe. Please see **webactions.html** and **using-webactions.html**.

6.1.2 Major new features in Allegro CL 7.0

The most significant addition in Allegro CL 7.0 is a new model for multiprocessing. See [Section 6.1.4 The new model for multiprocessing and OS threads in Allegro CL](#) for more information.

Other new capabilities to Allegro CL are listed here, with links to the documentation of the new features. Some of these features have also been released as patches to Allegro CL 6.2.

1. **Environments functionality:** the document [environments.htm](#) describes the environments functionality added to Allegro CL in release 7.0. Based on the environments proposal described in section 8.5 of *Common Lisp: the Language* 2nd ed. (but not in the event adopted by the X3J13 standards committee), the environments functionality allows programmers to better define the current environment (or the environment that will be in effect when an application is fully loaded) to the compiler.

7.0 note: environment support was added as a patch to Allegro CL 6.2. There have been some changes in the 7.0 release, particularly to **function-information** and **augment-environment**. 7.0 user familiar with the 6.2 version should look at the new descriptions.
2. **Maximum array sizes are larger, new short array type allows using the old arrays as well.** See [Arrays and short arrays in implementation.htm](#) for a definition of short arrays and a discussion of the array implementation in Allegro CL. The change means that type codes and the internal structure of arrays are now different. If your code depended on the old structure (if, for example, foreign code accessed and worked with Lisp arrays), your code may not work with the new arrays. In that case you can use the short arrays provided for this purpose. Also, the new larger (limit) arrays use an additional word of memory per array. If you are creating a very, very large number of (presumably small) arrays, this may be important and you may wish to use short arrays. Otherwise, there is no reason to worry about short arrays: just use the standard Common Lisp array functionality as always. Note that if you do use short arrays, most standard Common Lisp array function will not work (**aref**, **arrayp**, and **vectorp** are exceptions); you use the newly-provided short array functionality. Read [Arrays and short arrays in implementation.htm](#) carefully to understand how array code must be modified when using short arrays. Also see [Appendix G The effects of the new, longer array implementation on def-foreign-call and def-foreign-type](#) for important information on **def-foreign-type**.
3. **New Regular Expression API: the regexp2 module.** A new, fast, PERL-compatible regular expression matcher has been added to Allegro CL 7.0. It uses PERL syntax and is 30% faster than PERL on the CL-CCPRE test suite. Regular expressions can use the Unicode character set (UCS-2). The older regexp module remains available. You may use both at the same time in the same running Lisp. Both modules are described in [regexp.htm](#). The section **The new regexp2 module** describes the new API. (The older module, which has been available for some time, is described in section **The older regexp API**.)
4. **Allegro Prolog: an implementation of Prolog within Allegro CL.** Prolog is a backward-chaining, logical programming language. It is now implemented within Allegro CL. The file [prolog.html](#) provides the documentation. (The documentation does not describe Prolog itself: users are assumed to be familiar with Prolog.)
5. **New interface to Oracle databases.** A new interface to Oracle databases has been added to Allegro CL 7.0. The interface is described in [oracle-interface.htm](#). The interface is similar to the Allegro ODBC interface (see [aodbc.htm](#)) and the MySQL interface (see [mysql.htm](#)), but is specialized for Oracle databases. While you must use the Oracle C Interface Libraries, no ODBC library is necessary.
6. **Support for the Document Object Model.** The Document Object Module (DOM) provides a programmatic interface to XML. The DOM implementation is connected to the SAX implementation mentioned above (both are part of the [sax.fastl](#) module). The DOM implementation is described in [dom.htm](#).
7. **New encoding functions in addition to the existing MD5 facility.** For some time Allegro CL has supported MD5 encoding. In release 7.0, support for HMAC, RC4, and SHA1 encodings has been added. See [MD5, SHA1, HMAC, and RC4 support in miscellaneous.htm](#). Note that the interface to the MD5 support function has been changed in release 7.0.
8. **Additions to SWIG provided by Franz Inc. allow automatic generation of def-foreign-call forms.** SWIG is a software development tool that reads C/C++ header files and generates the wrapper code needed to make C and C++

- code accessible from other languages. See <http://www.swig.org>. See http://www.franz.com/support/tech_corner/swig042804.lhtml for specific information on the interface to Allegro CL and information on downloading the software. (The SWIG software is not included with the distribution because it is regularly update. Users should always get the latest update.)
9. **DBM support in Allegro CL.** The `:ndbm` module defines various operators in the `dbi.ndbm` package providing support to the DBM database tools provided on UNIX machines. This is a simple database tool designed for simple, generally static data. The tools in Allegro CL access and manipulate these data, and have iterators to traverse the whole database. See [ndbm.htm](#) for more information. (This facility is not supported on Windows and currently does not work on HP 64-bit. It may be fixed on HP 64-bit with a later patch.)
 10. **Automatically inflating gzip-compressed files.** The `inflate` module contains tools for creating a stream class which will automatically inflate gzip-compressed streams as they are read. See [Support for gzip decompression in miscellaneous.htm](#) for details.
 11. **New interface to OpenGL.** 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. [../opengl/readme.txt](#) has a brief introduction to the interface. More information is in these files: on the Common Graphics veneer: [../opengl/cggl/doc.txt](#) (Windows only). More information is in these files: on the GTK veneer: [../opengl/gtkgl/doc.txt](#). (We also have an interface to GTK, mentioned above -- see [../gtk/readme.txt](#). The GTK veneer on OpenGL provides OpenGL additions in GTK style.)

6.1.3 Other new features in Allegro CL 7.0

1. ***additional-logical-pathname-name-chars*** allows arbitrary characters in logical pathname namestrings. As specified by the ANSI spec, words in logical pathnames must consist of alphabetic characters, decimal digits, and the minus sign. Not allowed are characters such as underscore (`#_`). Users have found this restriction onerous and unnecessary. The value of a new variable, `*additional-logical-pathname-name-chars*` should be a list of character objects. These characters will then be allowed in logical pathname words. (Logical pathnames containing characters in `*additional-logical-pathname-name-chars*` which are not alphabetic characters, decimal digits, or the minus sign will not be portable, of course.) The initial value of `*additional-logical-pathname-name-chars*` is `nil`. See the description of `*additional-logical-pathname-name-chars*` and **Logical pathnames: general implementation details** in [pathnames.htm](#) for more information.
2. **New condition `syscall-error`, now parent of `file-error`.** `syscall-error` is a subclass of `error`. An error of type `syscall-error` might be signaled when there are problems with functions that interact with the operating system such as `run-shell-command`, `shell`, and `chdir`. `syscall-error` has one slot, `errno`, with accessor `syscall-error-errno`.

As part of this change, the `file-error` is now a direct subclass of `syscall-error`, and only a subclass (but not direct) of `error`.
3. **New condition `winapi-error`.** `winapi-error` is a subclass of `error`. An error of type `winapi-error` might be signaled when there is a problem with the Windows API on Windows.
4. **setf method for `file-write-date`.** A `setf` method has been provided for `file-write-date`. It sets the modification time (the `mtime` in UNIX and the equivalent on Windows). See [cl:file-write-date](#) in [implementation.htm](#).
5. **New setf'able function `excl:file-access-date`.** The new function `file-access-date` returns (or with `setf`, sets) the last access (read or write, including write-like utilities like `touch`) time of the file.
6. **New command-line arguments `+N` and `+Ti` (on Windows) allow customization of tray menu items.** `+Ti` removes **Interrupt Lisp** from the tray menu. `+N appname` uses `appname` rather than **Lisp** in tray menu items. Thus `+N Myapp` will cause the menu item that is normally **Interrupting Lisp** to be **Interrupting Myapp** (and `+Ti` removes that item). See [Command line arguments](#) in [startup.htm](#) for information on command line arguments.
7. **New command-line arguments `+Cx` and `+Tx` (on Windows) make it harder for users to accidentally close a running application.** `+Cx` disables the Close button on the console window and `+Tx` disables the "Exit Lisp" menu

on the system tray. These arguments are useful for applications that run in the background (an NT service application) and perform some useful function (e.g. implementing an NFS server). See **Command line arguments in startup.htm** for information on command line arguments.

8. **New variable `*system-messages*` controls where load and autoload are sent.** The value of the new variable `*system-messages*` should be an output stream, `t`, or `nil`. It controls where messages generated by **load** and by system autoloading are sent. Autoloading messages are now controlled by `*load-verbose*`, just as **load** messages are. If `*load-verbose*` is `nil`, no messages are generated by successful loads or by autoloads. (In earlier releases, it was difficult to suppress autoloading messages.) See **Autoloading in implementation.htm** for more information on autoloading.
9. **New function `console-control` (Windows only) allows control over the console, system tray icon, and certain gestures.** The function `console-control` only works if Lisp or a Lisp application is started with the Lisp console (i.e. the `+c` command-line argument is not specified). The function will display or minimize or hide the console, display or hide the system tray icon, and specify the action taken when the console close button is clicked.
10. **New variable `*pathname-customary-case*` specifies the customary case for pathname functions.** When the `:case` keyword argument to **make-pathname**, **pathname-host**, **pathname-device**, **pathname-directory**, **pathname-name**, and **pathname-type** is specified `:common`, the case of components is converted according to the value of `*pathname-customary-case*`. We recommend that users not specify a value for `:case`. If `:case` is unspecified, Allegro CL behaves as it did in previous versions.

6.1.4 The new model for multiprocessing and OS threads in Allegro CL

Processes within Lisp are now instances of a CLOS class that can be specialized to get customized behavior. Process objects mediate the programmer's access to threads. The programmer creates an instance of the process class (or of a subclass of the process class) in order to start a parallel computation. The computation's state can be inspected and modified through the use of functions that receive the process object as an argument.

Because processes are CLOS instances, the programmer can create specialized process classes whose instances contain application-specific attributes or behaviors. A server application, for example, could add slots to hold request queues or sockets or transaction-status data structures. After methods defined on **initialize-instance** and **process-terminate** could perform application-specific startup and shutdown actions.

This new model allows greatly simplified system design and implementation. It allows the programmer to treat the process and the server instance as a single object, instead of forcing a program structure in which they are two distinct things, one of which owns the other.

Stack groups no longer supported

The following functions are no longer defined:

- **make-stack-group**
- **profile-stack-group-p**
- **stack-group-funcall**
- **stack-group-name**
- **stack-group-p**
- **stack-group-preset**
- **stack-group-resume**
- **stack-group-resumer**
- **stack-group-return**
- **stack-group-state**

- **process-stack-group**: use **process-thread**.
- **syneval-in-stack-group**

Other removed functionality

- **make-immigrant-process**
- **process-implementation**: this macro expanded into a call to **process-thread** on `:os-threads` implementations and **process-stack-group** on non-`:os-threads` implementations. In 7.0, **process-thread** can be used in either model. Calls to **process-implementation** can be changed to calls to **process-thread**.
- **process-flush**. Use **process-kill**.
- **excl:lisp-sleep**. **cl:sleep** is now effectively `process-sleep` always.

New functionality

- **process-alive-p**
- **process-cpu-msec-used**
- **process-cpu-msec-used-delta**
- **process-keeps-lisp-alive-p**
- **process-os-id**
- **process-progn**
- **find-process**
- **process-terminate**
- **process-times-resumed**
- **process-times-resumed-delta**
- `mp:*general-wait-delay*`
- **process-message-interrupt-function**

Other notes

- There can be only 350 simultaneously running processes. See **The maximum number of simultaneous processes in multiprocessing.htm**.
- `*current-process*` will be non-`nil` if the process code has been loaded, either during the build of the lisp image or by `(require :process)`.
- **start-scheduler** now simply initializes some `mp` structures in both models. In the native-thread model, it opens the doors to immigrant threads.
- **make-process** now accepts a keyword argument `:class` to specify the class of the process to be created. This defaults to `mp:process`, but may be specified as an application-defined subclass of `process`.
- **process-resume-hook/process-suspend-hook**: note that in the native-threads model, a process can release the heap for another process to run even though the first process is not truly suspending. A foreign call can release the heap, after which it may block, e.g., in a `GetMessage` call in windows, or continue to process in the background outside the lisp heap. In the native-threads model the `suspend-hook` is called whenever a process releases the heap and the `resume-hook` is called whenever it reacquires the heap.
- **process-run-function** and **process-run-restartable-function** accept a keyword `:class` in the *name-or-keywords* argument to specify the process class to create (default being `mp:process`).
- **process-reset** function will signal an error if the *kill* argument is non-`nil`, or if the process is an immigrant. The latter can only happen in a native-threads lisp.
- **process-kill** now accepts a keyword argument `:wait`. If non-`nil`, the calling process waits until the killed process is really gone. `process-kill` signals an error if the process to be killed is an active immigrant. An inactive immigrant is

one that was created to handle a lisp call from a foreign thread, and has returned from the topmost lisp call back into the foreign regime. The thread may still be processing, but it has no lisp state. This will kill the lisp process associated with that foreign thread, but will not kill the foreign thread itself. If it later calls into lisp, a new immigrant process will be created for it.

- **process-interrupt**: processing an interrupt function can be interrupted by additional process interrupts that occur before the current one has finished executing.
- **start-customs**: Should be replaced by a call to **start-scheduler**. The current version warns and calls **start-scheduler** when it is called unless the scheduler is already running. This function is here for compatibility and will be going away some day.
- **with-timeout**: This is now also **sys:with-timeout**, and is available even when the process module is not loaded.
- **process-allow-schedule**: the optional *process* argument is ignored.

6.2 Non-backward-compatible changes in the base Lisp

1. **simple-string, single-float, and double-float functions removed**. These symbols, which name classes in Common Lisp, also named functions in earlier releases of Allegro CL (essentially doing `(coerce x '[name])` or `(string x)`). Using these names for non-standard CL functions was never legal according to the Common Lisp standard. These functions have now been removed. Calls to the **single-float** or **double-float** functions should be replaced with calls to **coerce**, as in this example:

```
(double-float 3) should be changed to (coerce 3 'double-float)
(single-float val) should be changed to (coerce val 'single-float)
```

If the compiler encounters calls to these function while compiling code, a warning of class `incompatible-conformance-change-warning` is signaled.

2. **extloop.cl and oldloop.cl removed from the distribution**. In releases prior to 7.0, the files *extloop.cl* and *oldloop.cl* were provided in the *src/* subdirectory of the distribution directory. These provided compatibility with older versions of the **loop** macro. They have been removed in release 7.0.
3. **Allegro Presto facility removed**. This facility, which was a space-saving option in earlier releases is no longer supported. Since use of Allegro Presto was transparent (code worked the same whether or not the facility was enabled), its demise should not affect user code. However, in certain cases specifying the use of Allegro Presto (for example, to **load** or to **build-lisp-image**) will cause a warning to be signaled. See **The Allegro Presto facility has been removed in loading.htm** for more information.
4. **The profiler has been renamed the runtime analyzer**. *Runtime analyzer* more accurately describes what it does. The interface and API remain the same, except that the names of various IDE dialog and menu items have been changed.
5. **Command-line argument functions ignore + arguments**. This has in fact always been true but was not previously correctly documented. The functions that retrieve command-line arguments in a running image, **command-line-argument**, **command-line-arguments**, and **command-line-argument-count**, ignore + arguments and return no information about them. The + arguments are for Windows only and modify the Windows startup behavior of Allegro CL and generated applications. These arguments are transient in nature (e.g., one calls for the program to start minimized, while another suppresses the splash screen) and as such are not reflected in the command line arguments available from Lisp. **Command line arguments** in **startup.htm** lists all command-line arguments accepted by Allegro CL, including the + arguments.
6. **Wait functions used by mp:process-wait may be called many times and may be called after they have already returned true**. **mp:process-wait** determines when it should stop waiting by calling the wait function specified in the **mp:process-wait** form. This wait function may be called an arbitrary number of times and may be called after it has returned non-`nil` (that is, after it has already indicated that the waiting should stop). Because of this, wait functions with side effects should be coded very carefully.

It has always been true that wait functions could be called often and called after returning non-`nil`. However, multiprocessing changes in 7.0 mean that calling after returning non-`nil` is much more common, so wait functions with side effects that seemed to work okay in earlier releases may now cause problems.

See the description of `mp:process-wait` for more discussion of wait functions and suggestions for efficient ones.

7. **Format of value for `:application-administration` argument to `generate-application` changed.** The value of the `:application-administration` keyword argument to `generate-application` (actually fully documented in [Creating the deliverable in `delivery.htm`](#)) is a list of elements of the form `(type-keyword ...)`, where `(type-keyword ...)` could be (in earlier versions) `(:resource-command-line "... command-line args ...")`, which worked on UNIX only. Now `(type-keyword ...)` can be of the form `(:resource-command-line "arg1" "arg2" "arg3" ...)` (i.e. each argument in its own string) and this works on both UNIX and Windows.
8. **New function `function-name-p` tests whether argument is a suitable argument for `fboundp`.** As described in [Appendix A.1 Conformance fixes which might break existing code](#), Allegro CL 7.0 has a number of changes compared to earlier releases to bring it into compliance with the ANSI specification in various areas. Some of these changes cause Allegro CL to signal errors where before no error would be signaled. One such change is that only valid function names are accepted as arguments to `fboundp`. In earlier releases, `fboundp` returned `nil` rather than signaling an error when passed an argument which was not a valid function name or specification. (Thus, e.g., `(fboundp 3)` returned `nil` rather than signaling an error.) The new function `function-name-p` returns true if its argument is a valid function spec (and thus a suitable argument to `fboundp`) and returns `nil` otherwise. `(if (function-name-p spec) (fboundp spec))` thus behaves in release 7.0 as `(fboundp spec)` did in earlier releases.
9. **`:case` argument to certain pathname functions no longer ignored.** In releases before 7.0, Allegro CL did not implement pathname case, and the `:case` keyword argument to the `make-pathname`, `pathname-host`, `pathname-device`, `pathname-directory`, `pathname-name`, and `pathname-type` was accepted but ignored. Now `:case` is no longer ignored. See `*pathname-customary-case*`.
10. **Fix to `subseq` means it will now signal error where it did not before.** The form `(subseq "a" 0 4)` asks for a subsequence containing the first four elements of the sequence (string) "a", which, of course, has only one element. In earlier releases of Allegro CL, no error was typically signaled when the argument was a vector. Instead, a vector with garbage elements was returned. Now in 7.0 (and in 6.2 with a patch), an error is signaled. This may break code which improperly relied on the no-error behavior. The condition associated with the error is `simple-error`. The error message says **Error: In `subseq`, `end' (4) is beyond the end of the sequence (1).**, with `(4)` -- the value of `end`, and `(1)` -- the length of the sequence -- being replaced with the correct values for the call. See `subseq`.
11. **Potential problems with foreign functions returning `:int`.** `def-foreign-call` is defined to default its `:returning` keyword argument (specifying the expected type of value returned by the foreign call) to `:int`. `:int` corresponds to C's `int` type. However, if the foreign function does not actually return an `int`, subtle bugs could be introduced in programs, particularly if the C function returns a `long`, an `unsigned long`, or a pointer of some sort. In 32-bit Lisps, returning those values is not a problem (when `:returning :int` is specified or defaulted to) because `int` is always 32 bits on every architecture we support. But on 64-bit Lisps, if a 64 bit value is returned, the upper 32 bits are lost. If the value was not correctly sign-extended by the foreign code, a negative value in the foreign code could be seen by Lisp as a large positive value. A bug fixed by a patch in release 6.2 makes Lisp sign-extend values returned by foreign code in 64-bit Lisps, and thus values expected by the pre-patch behavior in 6.2 will now be different. So, on 64-bit Lisp, use `:returning :unsigned-long` when the return value is some kind of pointer. When it is an integer value, be sure to use the correct type and be sure that the foreign code actually produces that type.
12. **Modified example in `add-signal-handler` makes clear `signal()` should never be called directly.** The example in the description of `add-signal-handler` in earlier releases called the `signal()` function. This was a mistake and following the example could cause Lisp to fail on some platforms. User code should never call `signal()` but should call `lisp_signal()` instead. The corrected example calls `lisp_signal()`.
13. **When Lisp is started with a script: initialization files are not read; scripts that signal an error will exit with a non-zero exit status; new `-#T` command-line argument.** As described in [Starting on UNIX using a shell script in `startup.htm`](#), you can on UNIX start Lisp with a shell script. When you do this, no initialization files are read, as if Lisp was started with the `-qq` command-line argument. In earlier releases, initialization files were read and there was

no way to prevent them from being read.

Now when a script run using `#!` signals an error, it will exit with a non-zero exit status.

The new `-#T` command-line argument for scripts is like the existing `-#C` argument 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.

14. **New compiler style warnings.** `compiler-inconsistent-name-usage-warning` is signaled when a tag or variable is unused (and the variable is not declared ignorable) or a variable is declared ignore but used. `compiler-unreachable-code-warning` is signaled when code cannot be reached (i.e. a clause after the `t` or otherwise in **cond** clause or a **case**).
15. **New compiler switch `verify-funcalls-switch`.** If `verify-funcalls-switch` is false, compiled code will be faster because the jump will be directly to the **funcall**'ed function's start address. Debugging and runtime analysis will be impaired, however.
16. **The `-H` command-line argument removed.** This argument was supposed to allow you to specify a directory to be the translation of the `sys:` logical host (the directory containing the Lisp executable is used by default). However, the argument did not work consistently and has been removed. See **Command line arguments** in **startup.htm** for a list of command-line arguments accepted by Allegro CL.
17. **`apropos` now does case-insensitive search by default.** `apropos` finds string pattern embedded in symbol names. Allegro CL has for some time supported an additional optional argument, `case-insensitive`, which controlled whether the matching was case-insensitive or not. That argument now defaults to `t`, meaning case-insensitive matching be default. (In earlier releases, `case-insensitive` defaulted to `nil`.)
18. **`equal` on pathnames on Windows ignores case.**
19. **Minor change in `(namestring (pathname "."))`:** it returns `"/` instead of `."`.
20. **`open`, `probe-file`, `truename`, and `rename-file-raw` have new `follow-symlinks` argument, `(delete-file (truename p))` deletes the actual file rather than the symbolic link when `p` is a symbolic link.** `truename` now resolves symbolic links to the canonical filename. In earlier releases (until a 6.2 patch), it returned the symbolic link when presented with an argument that evaluated to a symbolic link. **`probe-file` and `open`** (when called with `:direction :probe`) return the `truename` when their file argument exists. The new argument (which defaults to `t` for `truename`, `open`, and `probe-file` and to `nil` for **`rename-file-raw`**) causes **`truename`** and the functions that call it (as all the others do) to resolve the symbolic link when true and to return the symbolic link when `nil`. This means that `(delete-file (truename p))` deletes the actual file (prior to the 6.2 patch, it deleted the symbolic link only). `(delete-file (truename p :follow-symlinks nil))` deletes the symbolic link. See **`rename-file-raw`** and **Extensions to `cl:make-package`, `cl:disassemble`, `cl:truename`, `cl:probe-file`, `cl:open`, `cl:apropos`** in **implementation.htm**
21. **`ipc` module removed.** Use the socket interface instead (see **socket.htm**).
22. **directory given wildcards like `*/*.cl` will ignore files which are symbolic links that point to other directories.** This prevents **`directory`** recursing into this symbolically named directories. For example, `(directory "*/*.cl")` will no longer, in the face of ``foo'` symlink to a directory, would descend into ``foo'`.
23. **Various defining forms no longer persistent after compile file.** Defining forms like **`defmacro`** and **`defstruct`** can appear in a file compiled by **`compile-file`**. The question is, do the definitions persist in the running image after file compilation completes (absent **`eval-when`** instructions). In earlier releases, the persistence behavior was inconsistent (**`declaim`**, **`defpackage`**, **`defsetf`**, **`define-symbol-macro`**, persisted and **`defvar`** and **`defparameter`** partially persisted, but other defining forms did not). In 7.0, only **`defpackage`** persists. This means that code which depended on persistence of particular defining forms (typically in a second file to be compiled) will have to be modified. The typical modification is to wrap the defining forms which need to persist in a `(eval-when (compile load eval) ...)` form.

It is likely the most common whange will invlove `defvar` and `defparameter`, because the associated special declamation which previously persisted no longer does. Thus the compilation of the file `foo1.cl` after compilation of `foo1.cl` signals a warning in 7.0 but did not in earlier releases:

```

;; file foo1.cl begin
(in-package :user)

(defvar *foo1* nil)
;; file foo1.cl end

;; file foo2.cl begin
(in-package :user)

(defvar *foo2* nil)

(defun bar ()
  *foo1*)
;; file foo2.cl end

cl-user(1): :cf foo1
;;; Compiling file foo1.cl
;;; Writing fasl file foo1.fasl
;;; Fasl write complete
cl-user(2): :cf foo2
;;; Compiling file foo2.cl
; While compiling bar:
Warning: Free reference to undeclared variable *foo1* assumed special.
;;; Writing fasl file foo2.fasl
;;; Fasl write complete

```

The warning will not occur if either the defvar is wrapped in an eval-when -- (eval-when (compile load eval) (defvar *foo1* nil)) -- or a declaim form is added and wrapped in an eval-when -- (eval-when (compile load eval) (declaim (special *foo1*))).

See **Persistence of defining forms encountered by compile-file in compiling.htm** for further details.

24. **device-open now has three required arguments instead of two.** In release 6.2, **device-open** had two required arguments, *simple-stream* and *options*. In 7.0, it has three. The *initargs* argument is essentially a new name for the old *options* argument. **device-open** now is similar to **shared-initialize**. (Programmers write methods for device-open when doing advanced stream work, so your code sees the arguments passed. Programmers do not call device-open directly. **device-open** methods must be modified because of the change in the number of required arguments.)
25. **device-extend is no longer supported.** Methods on **device-extend** should be changed to methods on **device-read** or **device-write**. See the description of **device-extend** to see how to ensure your application does not have any lingering **device-extend** methods.
26. **New simple-stream strategy functions j-read-byte and j-write-byte.** See **Strategy descriptions necessary for encapsulation in streams.htm**.
27. **float-declaration-used-warning condition removed.** The condition class named by the symbol `excl:float-declaration-used-warning` has been removed. In earlier releases, a warning was signaled when a variable was declared a float, rather than a single-float or a double-float. Because Allegro CL supports to floating-point types, declaring a variable to be a `float` could result in less than optimal code, because inlining of mathematical routines was not possible. However, there are other reasons to declare a variable to be a float, for example when methods are written on the float class and the program is telling the compiler that those methods apply to a variable. Warnings are inappropriate in that case, but (except for inconvenient additional programming) cannot be avoided. The `:explain` declaration can be used to ensure floating-point code is properly optimized. See **Help with declarations in compiling.htm** for further details on optimizing floating-point code.
28. **Functions called by the :call option to search lists are now passed five rather than four arguments.** The new fifth argument is `check-lower-case`, which will be non-nil if there are no lowercase characters in the target filename (in which case, trying the lowercase version of the target might be desirable). Code which uses functions and the `:call`

option to search lists must be revised so these functions accept a fifth argument. The two supplied search lists are `*load-search-list*` and `*require-search-list*`. See **Search lists in loading.htm** for information about search lists.

29. **MD5 functions have changed interface.** The functions **md5-update**, **md5-final**, **md5-string**, and **md5-file** all have new argument lists. **md5-update** now accepts keyword arguments rather than a single optional argument (the optional argument was *len*, specifying how many characters from a string to operate on, the keyword arguments are *start*, *end*, and *external-format*). A three-argument call to **md5-update** is still accepted, with a warning.

md5-final, **md5-string**, and **md5-file** all had a keyword argument `:usb8-result` added with a 6.2 patch. The argument has been changed to `:result` (with values `:integer`, equivalent to `'usb8-result nil'`; `:usb8`, equivalent to `'usb8-result t'`; and `:hex`, causing a string with a hex representation of the result to be returned). The `:usb8-result` argument is no longer accepted. **md5-final** now has additional keyword arguments which are passed to **md5-update**. **md5-string** and **md5-update** now accept (unsigned-byte 8) vectors as well as strings as arguments.

See **MD5, SHA1, HMAC, and RC4 support in miscellaneous.htm**.

30. **new-start-emacs-lisp-interface now start the interface on the Lisp side.** The Lisp connection to the emacs-lisp interface is now made by the function **new-start-emacs-lisp-interface**. The obsolete older function **start-emacs-lisp-interface** is still supported to handle starting the connection with an old version of the interface. The change only affects the situation where you are starting the interface after the Lisp itself has started. (The usual way of starting the interface, with the emacs function **fi:common-lisp**, uses the correct Lisp-side function automatically.) If you are making the connection in such a fashion, you must use the new function instead of the old except in one anomalous situation: you are running Allegro CL 7.0 but using the 6.2 interface.

6.3 Other changes to and notes about the base Lisp

1. **New pprint inspector option.** The **:istep** top-level command (for navigating through inspected objects once the inspector has started) now has the option `pprint`. It causes the current object to be pretty printed. It is useful when multiple options are specified in a single **:istep** command, since `intermediate` is not printed. Interspersing `pprint's` among the specified options causes `intermediate` values to be printed.
2. **dns-query now accepts a search keyword argument.** (This change was introduced as an Allegro CL 6.2 patch.) **dns-query** now accepts a search keyword argument. It defaults to `nil`. If specified `true`, `*dns-domain*` and `*domain-search-list*` will be used to fully qualify the name argument before doing the query. **dns-lookup-hostname** (which simply calls **dns-query**) accepts the *search* keyword arguments since it accepts all **dns-query** keyword arguments.
3. **New variable controls what kind of socket is created for the Emacs-Lisp interface.** The value of the newly exported variable `*eli-daemon-socket-hook*` should be a function that accepts one argument and will initialize the socket that is used to communicate with Emacs when the Emacs-Lisp interface is started. See the description of `*eli-daemon-socket-hook*` for details.
4. **Changes to delete-directory-and-files.** This change was made in an OSI patch released for 6.2. **excl:delete-directory-and-files** changes: (1) add *force* keyword which causes read-only files on Windows to be forcibly removed (this is the default behavior on UNIX), (2) change the default of the *quiet* keyword to `t`, (3) change the default of the *if-does-not-exist* argument to `:error`, and (4) have the files removed in the proper order, so that subdirectories are emptied before their parents.
5. **New :always-append value for :if-exists and :if-does-not-exist arguments to open.** This change was made in an OSI patch released for 6.2. The *if-exists* and *if-does-not-exist* keyword arguments of **cl:open** now accept the value `:always-append`, which causes **O_APPEND** to be used when opening the file. (This only has meaning for *if-does-not-exist* when the file is opened for output.) This means that concurrent writes by any number of programs will always write to the end of the file. This is useful for writing to log files. Be warned, however, that changing the file position of a stream opened with `:if-exists :always-append` or `:if-does-not-exist :always-append` will have no effect. See the description of the implementation of **cl:open** in **Extensions to cl:make-package, cl:disassemble, cl:open, cl:apropos in implementation.htm**.

6. **Changes to run-shell-command.** `excl:run-shell-command` now accepts a *directory* keyword argument, the directory in which the command is run. It also accepts *uid*, *gid*, *effective*, and *initgroups-user* arguments on UNIX platforms.
7. **Changes to copy-file.** This change was made in an OSI patch released for 6.2. `sys:copy-file` changes: (1) add *force* keyword which causes the file to be removed if there is an error opening it, and (2) add *remove-destination* keyword argument, which causes all files to be removed, if they exist, before the copy is performed. A new *verbose* keyword argument has also been added.
8. **excl:make-directory error reporting.** This change was made in an OSI patch released for 6.2. `excl:make-directory` now properly reports a `file-error` condition, with proper `errno` information when an error occurs.
9. **reap-os-subprocess now returns a third value: the number of the signal that caused the process to exit on UNIX/Linux platforms and nil on Windows platforms.** When a process is reaped, the first two returned values are the exit status and the pid. An exit status of 0 does not by itself mean the process exited normally: the number of the signal must also be checked. This third return value allows this on UNIX and Linux. In cases where no process was actually reaped, the third returned value is `nil`. On Windows where signal information is not available, the third return value is always `nil`. See `reap-os-subprocess` for details.
10. **New apropos-regex function allows using regular expressions in apropos.** `apropos-regex` is an extension of `apropos` that interprets its first argument as a regular expression and uses it to perform the search of symbols.
11. **New :force-compile defsystem long form module option.** Specifying this option forces compilation by `compile-system`. See `Long form module-specifications` in `defsystem.htm` for a list of the long form module options.
12. **Defsystem functions named by excl symbols have names exported from defsystem as well.** Various `defsystem` functions have always been named by symbols in the `excl` package. These have now also been exported from the `defsystem` package. They include `load-system`, `compile-system`, `map-system`, `show-system`, `concatenate-system`, `touch-system`, and `clean-system`.
13. **New format for the :explain declaration; new :tailmerging :explain quality.** The format of the `:explain` declaration in earlier releases was `(:explain :quality | :noquality)`, thus `(declare (:explain :boxing :nocalls :notypes :variables))` enabled explaining for boxing and variables and disabled it for types and calls. That format is still supported, but its use is deprecated. Instead, use of the new format, `(:explain :quality | (:quality t) | (:quality nil))` is recommended. `(declare (:explain :boxing (:calls nil) (:types nil) (:variables t)))` has the same effect as the old-style form above.

There is a new `:tailmerging` quality for `:explain`. It explains why a function in the tail position is or is not tail merged.

See `Help with declarations in compiling.htm` for more information.

`(:explain :types)` also now provides environment information. See `Explain types and calls in compiling.htm`.

14. **New +<number> command-line argument for Windows allows specifying the maximum console size.** The size essentially controls the amount of text that can be displayed in the console. The argument is ignored on Windows 98 and Me, where the maximum size is 25,000 bytes and cannot be changed. On other Windows platforms, the default size is 100,000 bytes and this argument can specify a different value. On those platforms, the value can also be changed after startup with `console-control`. See `Command line arguments` in `startup.htm` for a list of command-line arguments accepted by Allegro CL.
15. **The new function normalize-type takes a type specifier and returns a usually simpler type specifier.** The function `normalize-type` is useful when specifying types for, say, arguments, when the argument may be one of various possible types.
16. **Various function for determining locations, malloc'ing space, and freeing space.** `lispval-to-aligned-malloc-address` and `lispval-to-address` provide information about the location of lisp values. `acmalloc-aligned` and `acfree-aligned` allocate and free space in the C heap but use aligned addresses.

The new functions `excl:malloc` and `excl:free` are direct links to the system `malloc()` and `free()`. Note that space

allocated with **acmalloc** cannot be freed with **free** (you must use **acfree** or **acfree-aligned**) and space allocated with **malloc** cannot be freed with **acfree** or **acfree-aligned** (you must use **free**). Note further that **malloc** and **free** are different from the functions named by the symbols **excl::malloc** and **excl::free** in earlier releases. Those calls, if any exist in your code, must be changed to calls to **acmalloc** and **acfree**.

17. **Some functions with an address argument now have an aligned optional or keyword argument indicating address is an aligned address.** **char*-strlen** and **short*-wcslen** have an *aligned* optional argument. **string-to-native**, **native-to-string**, **octets-to-native**, and **native-to-octets** now have an *aligned* keyword argument. Functions like **lispval-to-aligned-malloc-address** return aligned addresses.
18. **Backward-compatible change to add-stream-instance-flags: flags can be zeroed as well as set.** If the first *flag-name* provided to **add-stream-instance-flags** is a list rather than a flag name, it should be a list of flag names and those are zeroed out before the remaining flag names, if any, are set. In earlier releases, this function could only set flags.
19. **Changes to room output: pure space information now included.** See the discussion of **cl:room** in **Getting information on memory management using cl:room** in **gc.htm**.
20. **Variable *load-foreign-types* lists suitable type extensions for foreign files; new :foreign keyword argument to load.** When a file has an extension listed in the value of **load-foreign-types**, then **load** will treat it as a foreign code file and act appropriately. The value is different on different platforms. To see the value of a platform you are using, evaluate this variable in a running Allegro CL image. The value should not be changed. You can, however, force a foreign load of a file with an extension not included in **load-foreign-types** by specifying a true value for the new (and non-standard) **:foreign** keyword argument to **load**. See **Using the load function in loading.htm** for details of the Allegro CL implementation to **load**.
21. **Information about user-defined top-level aliases now included in :help output.** Top-level alias definition can include a documentation string (like **defun**), and that string will be used by the **:help** top-level command when describing the new alias. **:help** will also list the alias along with system-defined top-level commands. See **tpl:alias**.
22. **New conditions file-does-not-exist-error and autoload-file-does-not-exist-error.** *file-does-not-exist-error* is signaled when **load** tries to load a file that does not exist and the **:if-does-not-exist** keyword argument to **load** is true. *autoload-file-does-not-exist-error* is signaled when the system tries to autoload a file that does not exist.
23. **Install wizard now detects whether user has correct privileges to install the application.** The install wizard is a tool useful for delivering applications on Windows. See **Installation of your application on Windows using the Install Wizard** in **delivery.htm**.
24. **On Windows, sys:update-allegro now asks if you want to reboot immediately when a reboot is necessary.** See **sys:update-allegro**.
25. **Bug in chdir fixed.** A bug where **chdir** did not accept a pathname argument has been fixed. The argument to **chdir** can be a pathname or a string.
26. **Use of pathname-sans-file is deprecated; use path-pathname.** **path-pathname** and **pathname-sans-file** do essentially the same thing. Programmers should use **path-pathname**.
27. **New OSI module functions with-os-open-file and stat-type.** The **with-os-open-file** macro works like **cl:with-open-file** except it uses **os-open** instead of **cl:open** to open the file.

The function **stat-type** returns a keyword describing the type of file object returned by functions like **stat**.
28. **generate-application bug fixed.** **generate-application** now properly replaces old Allegro CL shared libraries, and other files, if an existing directory is used.
29. **:ssl-support is on *features* for platforms that support SSL.** SSL is discussed in **Secure Socket Layer (SSL)** in **socket.htm**.
30. **Emacs-lisp interface changes.** There is better support for multiple emacs/lisp instances on Windows (see **How to run two Emacs's connected to two different Lisps** in **eli.htm**; ELI startup on Windows no longer uses hardwired port ranges; and **start-emacs-lisp-interface** no longer ignores the *announce-to-file* argument on Windows.
31. **Trace output on Windows goes to the Emacs/Lisp interface listener if there is one.** In previous versions, output sent to **trace-output** went to the console.

32. **fasl files can be appended to #! scripts.** See **Starting on UNIX using a shell script** in **startup.htm** for information on using scripts to start Allegro CL.
33. New functions `string-to-base64-string` and `base64-string-to-string`. The functions **`string-to-base64-string`** and **`base64-string-to-string`** add to the base64 support described in **Base64 Support** in **miscellaneous.htm**.
34. **New function `sys:lispval-storage-type` replaces `sys:pointer-storage-type`.** Calls to **`pointer-storage-type`** can be changed to calls to **`lispval-storage-type`** simply by changing the operator name. **`lispval-storage-type`** returns more specific information. It also accepts an optional argument which allows a list of suitable types to be returned.
35. **Unadvising a `fmakunbound`'ed symbol is an error.** This entry just notes that it is an error to **`unadvise`** functions named by symbols that have been **`fmakunbound`**. The consequence of doing so are undefined (but in some circumstances the result may be the symbol-function of the symbol is re-established). The fwrapper facility signals an error in this case. See **fwrappers-and-advice.htm**.
36. **New function `prefix`.** **`prefix`** takes two sequences as arguments and returns non-`nil` if the second starts with the first. The return value, when true, is the index in the second where the first ends (so `(prefix "foo" "foobaz")` returns 3). regular-expression matching (see **regexp.htm**) can do similar testing, this function is easier to use for very simple cases.
37. **New functions `install-string-input-character-strategy` and `install-string-output-character-strategy`.** The functions **`install-string-input-character-strategy`** and **`install-string-output-character-strategy`** replace the now-deprecated **`install-string-character-strategy`**.
38. **New `:include-locales` argument to `generate-application`.** If this argument is non-`nil`, the contents of the `locales/` directory in the Allegro CL distribution are copied to the application directory. The size of the `locales/` subdirectory is about 2 Mbytes. The argument is described in **Creating the deliverable** in **delivery.htm**, where **`generate-application`** is fully described.

6.4 Base Lisp platform-specific information

There are no entries at this time. Information may be placed here in documentation updates after the initial Allegro CL 7.0 release.

7.0 Release Notes for CLIM

The CLIM manual has been updated for the 7.0 release. These changes should be noted:

- **`clim-utils::dovector` now evaluates `:from-end` argument at run-time in all cases.** In earlier releases, the `:from-end` to argument internal CLIM function **`clim-utils::dovector`** was evaluated at compile-time, and so correct code was generated only in the case where the `:from-end` argument was a compile-time constant. In 7.0, the argument is always evaluated at run-time. Because this function is internal, it is not supported and may be changed without notice. However, it is unlikely to change and CLIM users are known to use it. If you do use it, please be sure to note that fact in comments to your code so you know to ask if additional changes have been made if there seems to be a problem.
- **Only the region subclass of designs can be transformed, not the pattern subclass.** The introduction to section 7.4 of the CLIM manual talked, in earlier releases, about transforming designs. The actual implementation can only transform regions. Regions are a sub-class of design and pattern is another disjoint sub-class of design. There are no methods implemented to transform patterns. The documentation has been corrected.
- **`postscript-prologue` will emit integers for `%%BoundingBox`.**
- **`use-line-style` allows any line thickness.**
- On Windows, a problem where selecting the menu-bar could causes text-field input to be lost has been fixed.
- On Windows, a problem where a text-editor pane sometimes failed to recognize the ENTER keystroke when typing very fast has been fixed.

- On Windows, **make-pattern-from-pixmap** has been implemented.
- On Windows, a menu accelerator can be any keystroke, not just a letter.
- On Windows, all cursor shapes are supported.
- `:echo-character` functionality has been added to text-fields.

(Repeated from 7.0 Release Notes.) The documentation for CLIM is in an online PDF file, **clim-ug.pdf**. It has been updated since release 2.0 and is called *The CLIM 2.2.2 User Guide*. That document includes material formerly in the (printed) CLIM Release Notes.

On Linux and FreeBSD, Allegro CLIM works with Open Motif 2.1 and 2.2 (2.2 was released in early 2002). Open Motif 2.2 is available at no charge from www.openmotif.org. Allegro CL 7.0/Allegro CLIM will work with either version. Redhat Linux 7.2 is supplied with Lesstif, a version of Motif that **does not work with CLIM**. See **this faq item** for information on how to install Redhat Linux 7.2 without lesstif and how to uninstall lesstif if already present.

On Mac OS X 10.3, OpenMotif Motif 2.2.2 is required. CLIM will not work with version 2.2.0.

Certain CLIM demos on Solaris 64 bit give segv or otherwise fail when displaying over the network, language environment must be set to C. This is a problem when running any Sun GUI (such as the CDE environment or the Gnome 2.0 interface) on a Solaris64 machine. When bringing up the environment, set the language/locale to "C". (On the login screen, there's an "Options" button, which displays a menu that has a "Languages" submenu. Choose "C". Note: the default value is typically "en_US".) The "C" setting can process all of the 64-bit font sets. However, difficulties arise when displaying over the network. If you are displaying on the monitor of the machine running Allegro CL (and CLIM), the demos work as long as the language/locale is "C". They typically do not work when displaying on a monitor over the network. As of this writing, there is no fix for the problem.

8.0 Release Notes for Common Graphics and the IDE (Windows only)

Common Graphics and the Integrated Development Environment have been significantly revised in release 7.0. The main revisions are as follows:

- **New dialogs on startup:** when you start the IDE for the first time, you are asked about where you want project files to be stored. There are several dialogs that appear. These are not yet properly documented but their use should be fairly obvious.
- **Splitting off the dde and the ide packages from the common-graphics package.** DDE functionality has been moved to the `dde` module and the `dde` package. This allows you to use the DDE facility in an application without Common Graphics. See [Appendix D Details of separating DDE code from Common Graphics](#) for more details.

IDE functionality (that is, that part of Common Graphics used to develop applications and design components used in those applications) has been moved to the `ide` module and the `ide` package. The `ide` module will not be loaded into applications. See [Appendix B Details of Common Graphics/IDE package reorganization](#) for more details.

- **The CG classes have been reorganized.** Some have been renamed, some removed, and some new ones have been provided. See [Appendix C The reorganization of the CG class structure](#) for details.
- **Many obsolete functions etc. removed; others renamed.** Some of the removed functionality was deprecated in earlier releases. Some was never in fact implemented or was not useful in user code. A list of removed and renamed functionality can be found in [Appendix E Functionality/symbols removed or renamed in Common Graphics](#).
- **Finely dividing Common Graphics into independent modules.** This change is less user-visible: Common Graphics functionality has been divided into several dozen modules. Only those needed for an application have to be loaded in, thus saving space in the application. Whenever a module is loaded, all other modules on which it depends are also loaded. As part of this change, most every module has associated with it a specific package. These have names like `cg.icon` and `cg.tab-control`. All symbols are also exported from the `cg` package (and so there is no need to

use any of the new packages). You will occasionally see the packages referred to in descriptive output. `cg` is the parent package (in the sense of the Allegro CL hierarchical package naming scheme, described in **Hierarchical Packages** in [packages.htm](#)) of all the new packages. Note that `cg` is the proper name for what was previously called the common-graphics package. `common-graphics` is now a nickname.

- **The compatibility module for the old Common Graphics function names from the 3.0.2 release of Allegro CL on Windows is no longer supported.** Further, the `aclwin` and `aclffi` compatibility modules are no longer loaded by Common Graphics. See [Appendix F Common Graphics compatibility with aclpc 3.0.2 has been removed](#) for more details.

The [first subsection](#) describes changes to Common Graphics and the IDE that are non backward-compatible. Please review this section and make whatever necessary changes to your code to obtain the desired behavior in release 7.0.

The [second subsection](#) describes other changes to Common Graphics and the IDE. These should not require code changes (please tell us if any do, because that may indicate a bug), but note that certain function and argument names have been deprecated in favor of new names, and that new code should reflect these changes, and old code should be revised at some point.

The section [Section 8.3 IDE release notes](#) and its subsections provide information about the IDE.

8.1 Non-backward-compatible changes in Common Graphics

1. **Use of subclass-widget and subclass-widget unnecessary.** CG now internally calls `subclass-widget` on every `os-widget` instance. Applications can remove any calls to this function. Any leftover calls will now simply do nothing. `subclass-widget` is also unnecessary and should also be removed; calls to it too are now no-ops.
2. **set-focus now properly called in all cases.** We document that `set-focus` is always called when a window receives the keyboard focus, so that an application could add wrapper methods to track all focus movement. In fact `set-focus` was not called when interactively moving the focus to an `os-widget`. Now it is.
3. **A function now loads CG source file information, rather than a file.** In earlier releases, after obtaining the Common Graphics sources, you loaded the file `src/cg/loadsf.c` so the system would know where definitions of particular functions resided. In the new release, the file `src/cg/loadsf.c` has been removed and instead you call the new function `load-cg-source-file-info`. Also, IDE sources are now in `src/ide/`. CG sources are still in `src/cg/`. Source files are available to paying licensed customers, but you must sign a special license to get the key to decode the file provided on the distribution CD. Contact your Franz Inc. account manager for details.
4. **cursor-size** returns only two values. In previous releases, it returned four values but the third and fourth values were not useful.
5. The IDE no longer prompts for exit confirmation by default when the operating system is being shut down and there are no unsaved changes, since this was unusual behavior for an application and may have been annoying. The new IDE configuration option `query-os-exit` allows returning to the old behavior.

Methods may be added to the new generic function `os-exit-request` to determine whether an attempted shutdown of the operating system (or logoff of the current user) is allowed to proceed. The related function `query-end-windows-session` has been renamed to `os-exit-request` to avoid a Windows-specific name.

6. **Changed default for class argument to make-window.** The default value for the class argument to `make-window` has changed from `dialog` to `frame-window`. This change was made because the `dialog` class is not defined in the now-smaller base `cg` module (and because `frame-window` is a more fundamental type of window). Any call to `make-window` that did not pass a `:class` (or `:device`) keyword argument and that depended on the window being a `dialog` should now specify `":class 'dialog"`.
7. The name of the generic function `cg:display-windows-help` has been changed to `ide:display-help` (note the home package of the symbol naming the function has been changed from the `common-graphics` package to the `ide` package). See [Appendix B Details of Common Graphics/IDE package reorganization](#) for details of the move of symbols naming IDE-specific functions from the `common-graphics` package to the `ide` package.

8. The source code for the functions **new-text-editor**, **save-text-file**, and **save-as-text-file** has been changed. The descriptions of those functions includes the sources. (They are the default methods for the menu bar placed on a window by the IDE and the source is provided as a guide to writing your own method. The source for **open-text-file** is unchanged.
9. **update-dialog primary method should not be overridden.** In 6.2, a system-supplied `:after` method updated the item list of a dialog when **update-dialog** was called. In 7.0, this `after` method has been merged into the primary method. You should not override the primary method of **update-dialog**.
10. **do-default-restart no longer takes any arguments:** the function **do-default-restart** used to take a *console-state* keyword argument. It now does not accept any arguments.
11. **pop-up-message-dialog, pop-up-find-dialog, pop-up-replace-dialog, pop-up-string-dialog, and pop-up-strings-dialog** have been changed from generic to regular functions. Methods on them should be removed.
12. **cg:*edit-allowed-types* removed, some of its functionality replaced by new IDE configuration option.** The variable `cg:*edit-allowed-types*` has been removed. In the IDE, use the new ide configuration option **ide:file-dialog-source-types** instead. That variable used to supply the default for the `allowed-types` keyword argument to **ask-user-for-new-pathname** or **ask-user-for-existing-pathname**. Now a suitable default for that argument is supplied in the definitions of those functions. A value other than the default must be supplied as a value to that argument in the function calls.
13. **Auto-saving of project files no longer lists non-project files.** When the Save All dialog appears in the IDE for automatically saving unsaved changes, files that are not related to the current project are no longer listed and saved except when exiting the IDE, when all unsaved files are listed and saved as before.

When the Save All dialog appears in a situation where the current project is about to be closed (usually because you are opening another project or creating a new one), then the second button on the Save All dialog will say **Discard** instead of **No**, and selecting that choice will result in unsaved changes to the old project being lost. The button will also say **Discard** at IDE exit time to similarly warn that choosing it will discard all unsaved changes.

14. **Variable *check-cg-args* no longer exists, associated Run | CG Argument Checking menu command also removed.** The variable `*check-cg-args*` and the menu command **Run | CG Argument Checking** no longer exist. The IDE image used to contain a special version of Common Graphics that differed in various ways from the runtime Common Graphics fasl files, with one of the differences being extra code for checking argument types (when enabled). Now that a single version of Common Graphics is used, we decided against embedding this debugging code to avoid making standalone Common Graphics apps larger.
15. **property-editor-mode changed to property-editor-type, :editor-mode option/argument to defproperties, defcomponent, and define-property is now called :editor-type.** The `:editor-mode` option/argument to **defproperties**, **defcomponent**, and **define-property** is now called `:editor-type`. The corresponding generic function **property-editor-mode** is now called **property-editor-type**. This was to remove confusion with the IDE configuration option **editor-mode** (and to resolve a package conflict with a corresponding internal symbol).
16. **brief-comtab, emacs-comtab, host-comtab, and text-edit-comtab functions no longer exist.** The functions **host-comtab**, **emacs-comtab**, and **brief-comtab** no longer exist. These functions returned comtabs that are used internally in the IDE editor, and the IDE configuration option **editor-mode** (with its widget on the Editor tab of the **Options dialog**) is the exported way to select which comtab the editor uses. The code called by these comtabs exists mostly in the IDE only, and so these comtabs were not useful for applications. In addition, the function **text-edit-comtab** no longer exists. Instead, the IDE editor's comtab is the value of the variable `*text-edit-comtab*`, which could be modified to enhance the editor.
17. **Functions in windows package no longer exported from common-graphics package.** In releases prior to 7.0, these functions were named by symbols whose home package was the `windows` package but which were also exported from the `common-graphics` package. (These were documented with other `common-graphics` operators.) Starting in release 7.0, they are no longer exported from the `common-graphics` package. They are: **memory-status**, **os-version-info**, **file-systems**, **file-system-type**, **file-system-info**, and **file-systems-info**.
18. **Some functions moved from common-graphics to windows.** The home package of the following functions is now the `windows` package (in earlier releases, it was the `common-graphics` package). The symbols are not exported from the `common-graphics` package (as was done in earlier releases for `windows` package functions). The functions are: **network-machines**, **network-shares**, and **directory-subdirectories**.

- directory-subdirectories** has also been modified to accept a pathname namestring as well as a pathname object.
19. **Function eof-p removed.** The function **cg:eof-p** is no longer exported. It was IDE-specific and not useful for applications.
 20. **save-file changed: file argument is now required.** The *file* argument to **save-file** is no longer optional, and **save-file** no longer accepts an *allowed-types* keyword argument (which was not previously documented). When file was optional and not specified, a file choice dialog was displayed. An application should now first call **ask-user-for-new-pathname** when a path is needed, and use the user-chosen path as the value of the now-required *file* argument. See the compatibility note under **save-file** for more information.
 21. **top-level-dropper default changed from nil to t.** The default value of the **top-level-dropper** property of a *dropping-outline* widget has changed from `nil` to `t`. (The value `nil` provides a speedup, but one that is likely unnoticeable on modern machines.)
 22. **Function return-value-to-windows removed.** The symbol naming the function **return-value-to-windows** is no longer exported because that function is useful only internally within Common Graphics. Related functionality in Common Graphics for handling messages at the Windows API level had never been exported, so we expect that no Common Graphics applications have used this function.
 23. **Function system-color removed.** The symbol naming the function **system-color** is no longer exported. You can use the various functions such as **system-background-color** and **system-highlight-foreground-color** to obtain information on system colors.
 24. **Function clipboard removed, replaced by variable *clipboard*.** The function **clipboard** has been replaced by the global variable **clipboard**. The value of this variable is the stack of lisp clipboard values that had been returned by the function, which was an accessor of the **system** object.
 25. **reflect-texture-in-x and reflect-texture-in-y replaced by reflect-pixmap-in-x and reflect-pixmap-in-y.** **reflect-texture-in-x** and **reflect-texture-in-y** have been removed, replaced by **reflect-pixmap-in-x** and **reflect-pixmap-in-y**, respectively. The new functions take pixmaps rather than textures as arguments. The old functions that took a texture did not know the logical width of the texture, and would pad the end of the texture-to-reverse to match the internal array that is usually wider to meet a 32-bit boundary. The old functions also did not work at all if the internal array did not store one pixel per array element, which is often the case.
 26. **Argument to get-texture-info changed.** The first argument of **get-texture-info** (the *stream* argument) has been removed because it was not used at all. Any application calls to this function should be changed to no longer pass that argument.
 27. **stream-location function removed.** The deprecated function **stream-location** is no longer supported; call **owner** instead.
 28. **stream-device function removed.** The deprecated function **stream-device** is no longer supported; call **type-of** instead.
 29. **Variables erase, fill, invert, paint, and replace removed.** Use variables *po-erase*, *po-fill*, *po-invert* (or *po-xor*), *po-paint*, and *po-replace* instead. The symbols *erase*, *invert*, and *paint* have been removed. (*fill* and *replace* are common-lisp symbols and still exist, of course.)
 30. **check-fixnums function removed.** The deprecated function **check-fixnums** is no longer supported. This code does approximately what it did:

```
(defun my-check-fixnums (&rest args)
  (dolist (i args)
    (if (null (fixnum p i))
        (error "Non fixnum ~S passed to my-check-fixnums" i))))
```

31. **with-room macro removed.** This macro was not specific to Common Graphics. **cl:room** should be adapted to be used instead.
32. **outline-item function removed.** This deprecated function is no longer supported. Use **find-outline-item** instead.
33. **Functions list-to-tabbed-string and tabbed-string-to-list removed.** Use the more general base Lisp functions **excl:list-to-delimited-string** and **excl:delimited-string-to-list** instead.

34. **variable `*after-session-init-functions-hook*` removed.** Use `ide:*ide-startup-hook*` instead.
35. **button-p function removed.** It did not work correctly. Use `typep` instead.
36. **update-text-widget function removed.** Use `fetch-control-value` instead.
37. **web-page-contents function removed.** Use `net.aserve.client:do-http-request` instead (see [aserve/aserve.html](#)).
38. **pending-message function removed.**
39. **set-dialog-item-value and set-dialog-item-title functions removed.** Use `(setf value)` and `(setf title)`, respectively, instead.
40. **files-to-list-box, subdirectories-to-list-box, and pathname-string-from-directory-list-box functions removed.** Instead, simply call `cl:directory`, and use the returned values as the range of the item-list widget.
41. **pretty-printed-object function removed.**
42. **find-widget function removed.** Use `find-component` instead.
43. **compile-unsaved-form function removed.**
44. **application-type function removed.** Only one type of application (standard-exe) is currently supported by the project system.
45. **device-bitmap function removed.**
46. **cg:port-name changed to cg:printer-port-name; dde:port-name now documented properly.** The generic function `port-name` had methods for printer streams and for dde-ports. The method for dde-ports was not documented. The generic function and method for printer streams is now named **cg:printer-port-name**. **dde:port-name** is still the generic function with a method for dde-ports. The symbol `cg:port-name` has been removed.
47. **DDE variable `*show-dde-warnings*` replaced by `*generate-dde-messages*`.** The variable `cg:*show-dde-warnings*` has been removed, replaced by the variable `*generate-dde-messages*`. DDE messages are now passed to the new overridable generic function **dde-message**, allowing an application to handle or display the messages as desired. The DDE module has been separated from Common Graphics. See [Appendix D Details of separating DDE code from Common Graphics](#).
48. **Certain file types no longer in the default list used by the dialog displayed by File | Open command.** The IDE's **File | Open** command no longer includes specific choices for `*.lpr`, `*.bil`, and `*.bml` files in the **Files of Type** drop-down list of the file dialog, because these files are automatically generated and normally not edited in a text editor. But you can still specify (by entering the file name with the type, or choosing **All Files (*.*)** in the **Files of Type** box) and edit such a file as before if needed.
49. **Change in name of module that includes list-view code.** The optional code module that was called `:list-view-control` is now called `:list-view` to follow the convention where the name of the module for a control is the name of the control.
50. There is no longer a workaround in Common Graphics that is specific to Windows 95 (as opposed to all other versions of Windows including Windows 98) to avoid a particular bug with drawing pixmaps. If a pixmap's pixel array is defined in Lisp (see the **contents** property) and the top row of pixels is the first row of the contents list (see **invert-p**), and the pixmap does not have a "pixmap handle" (see **open-pixmap-handle**), then the pixmap may not always draw in Windows 95. To avoid this bug, you should either use **open-pixmap-handle** on the pixmap or else define the pixel array so that the bottom row of pixels is the first row of the contents list (setting the **invert-p** property of the pixmap to true). This bug does not affect pixmaps loaded from `.bmp` files, with **load-pixmap**.
51. **Project-related functions now in ide package are not available in applications but alternatives are available.** Some project-related functions no longer exist in a standalone Common Graphics app. You can no longer call **finder-function** or **maker-function** in a standalone app's **on-initialization** function (as **default-init-function** had done) to find the names of the auto-generated functions for recreating forms. Generally, project code should call the maker function or finder function directly, though **main-window-maker** is a new function that may be used in a standalone app.

load-project will work only in the IDE, not in a standalone CG application. To load code in a standalone app, simply use **load** to load `fasl` files.

52. ***starting-ide* is deprecated, use *ide-is-running*.** The variable `*starting-ide*` is deprecated in favor of the

more generally useful variable `*ide-is-running*`. `*ide-is-running*` is set to `t` when the IDE has started.

53. **Package for Common Graphics symbols is now `cg` rather than `common-graphics`, but `common-graphics` is a nickname.** The name of the Common Graphics package is now `cg` instead of `common-graphics` so that its new child packages such as `cg.list-view` will not have overly long names. `common-graphics` is retained as a nickname of the package for backward compatibility reasons, but its use is deprecated. It is recommended that `common-graphics: package` qualifiers in user code be changed to `cg:` in case the deprecated nickname is removed in the future.
54. **New `cuttable` and `pastable` properties of dialog items.** The new `cuttable` property of dialog-items return whether `cut-selection` and `delete-selection` will do anything on a particular widget. The new `pastable` property works similarly for `paste-selection` and `insert-selection`. `copy-selection` always works.

Cut and paste methods have also been added for item-list widgets.

55. **Changes to `copy-selection`, `cut-selection`, `delete-command`, `delete-selection`, `paste-selection`, and `insert-selection`; and the new function `insert-command`.** The functions `copy-selection`, `cut-selection`, `delete-selection`, `paste-selection`, and `insert-selection` may now be called on dialog-items in addition to windows. Previously you had to pass the `window` of the dialog-item.

However, these functions no longer pass the action down to the bottommost **selected-window**. These functions now instead act directly on the window or widget that is passed to the function. The former behavior was not documented, and it is not clear that all applications would want this particular indirection. (The convenience functions `copy-command`, `cut-command`, and `paste-command` do still pass the action in this way though.)

The functions `copy-selection` and `cut-selection` no longer return the window in which the copy or cut was done as the first returned value, since this is now always the window that was passed in (as noted above). Instead, the value that was copied or cut is now the first value returned instead of the second. And a new second value is returned, which is the clipboard format of the value that was copied or cut.

`paste-command`, `paste-selection`, `insert-command`, and `insert-selection` now return the pasted object and its clipboard format rather than undefined values. `insert-command` is new. `delete-selection` returns true if a deletion was done, and `nil` otherwise, rather than undefined values. `delete-command` is now deprecated because it simply calls `delete-selection`.

When `cut-selection` is called on a `rich-edit-pane` where no text is selected, `nil` is now returned rather than a rich text string that contains no actual (visible) text.

56. **Certain pixmap functions renamed for clarity, also no longer take an argument.** The following symbols have been renamed for clarity because they apply only to pixmap color vectors and not to palettes. (Palettes are associated with windows and not with pixmaps.) `initial-palette-vector` is now `initial-pixmap-color-vector`; `default-palette-vector` is now `default-pixmap-color-vector`; and `default-gray-palette-vector` is now `default-gray-pixmap-color-vector`. Any usages of those symbols in applications should be changed to the new names. The deprecated names still name functions for backward compatibility.

`initial-pixmap-color-vector` is identical to `initial-palette-vector`. However, both `default-pixmap-color-vector` and `default-gray-pixmap-color-vector` take no arguments while both `default-palette-vector` and `default-gray-palette-vector` take a single (*system*) argument. In fact, that argument is not needed and has for some time been ignored. The old functions still take that single argument. The new functions do not take any argument.

57. **palette-size methods for pixmaps removed.** The deprecated pixmap initarg `:palette-size` and the `palette-size` method for pixmaps no longer exist; for pixmaps, use the `:bits-per-pixel` initarg and the `bits-per-pixel` generic function instead. The function `palette-size` now applies only to `cg-streams`, to return the number of colors in the stream's palette.
58. **The window class associated with a new `lisp-widget` subclass should inherit only from `lisp-widget-top-window`, and not from `lisp-widget-window`.** When defining a custom `lisp-widget`, you create a window class for the window holding the widget. In earlier releases, this window class should be a subclass of both `lisp-widget-window` and `lisp-widget-top-window`. In 7.0, the only superclass should be `lisp-widget-top-window` (which is now a subclass of `lisp-widget-window`).

59. **list-view-cell-box now uses stream coordinates.** The box returned by **list-view-cell-box** is in the stream coordinates of the list-view, and so the value is not affected by the current scroll-position of the widget. Previously, the box was relative to the inner upper left corner of list-view's window. This change was originally made by a patch to Allegro CL 6.2.
60. **Default for on-line-segment-p is now 3.** The default for **on-line-segment-p** was 2 in earlier releases.
61. **copy-pixels-to-stream-from-file now creates a bitmap-stream rather than a bitmap-pane by default.** If **copy-pixels-to-stream-from-file** is called with no *stream* argument, the stream that is created automatically and returned is now a `bitmap-stream` rather than a `bitmap-pane`. This serves the same purpose without using a window that would not get shown anyway.
62. **result-rgb parameter of foreground-color and background-color has been removed.** The deprecated optional *result-rgb* parameter of **foreground-color** and **background-color** has been removed. The argument had not been used or needed in recent releases. Any application calls that pass this argument must no longer do so.
63. **Grid-widget function size renamed section-size.** Because of name conflicts with the package reorganization, the new name of what was the **size** function for grid-widgets is **section-size**.
64. **ignore-package-name-case configuration option has been removed.** This option was simply a wrapper for setting the global variable `excl:*ignore-package-name-case*`. The variable should be set directly. The symbol `ignore-package-name-case` was in the `cg` package but has been moved to the `ide` package. The symbol still has a function definition, but it is a no-op.

8.2 Other changes in Common Graphics and the IDE

1. **New functions for initializing and exiting Common Graphics allow using Common Graphics in development images not using the IDE and in applications not built using IDE tools.** The new function **initialize-cg** allows using Common Graphics in a Lisp that does not include the IDE, or a standalone app that was not created from an IDE project. Also new are **exit-cg** and `*cg-is-initialized*`. And the documentation for **standalone-application** has been rewritten.
2. **New macro in-cg-process simplifies testing Common Graphics code in a Lisp without the IDE.** The new macro **in-cg-process** provides a convenient way to run a body of Common Graphics code in a process that is set up automatically with the `*default-cg-bindings*` and a call to **event-loop**. It is particularly handy for testing Common Graphics code in a Lisp that does not have the IDE loaded.
3. **Use of cg:id deprecated.** Use of the function **id** is deprecated. The methods for `dialog-item` and `timer` should not be needed by applications, and **name** should be used instead for `header-info` and `tab-info`.
4. **New functions for controlling the splash screen of an application.** The new functions **kill-splash-screen** and **kill-splash-screen-when-ready** offer increased control over when the splash screen for a standalone Common Graphics application goes away at startup time.
5. **New functions for manipulating the tray icon and the console window in applications.** The new functions **show-console**, **hide-console**, **console-is-visible**, **console-title**, **console-tray-icon**, **console-tray-tooltip**, and **console-handle** allow a standalone application to manipulate the Allegro console window and its tray icon.
6. **New tray-item class with methods for custom tray items.** The new `tray-item` class and the new generic functions **add-tray-item**, **remove-tray-item**, **in-tray-p**, **tray-item-icon**, **tray-item-tooltip**, and **tray-item-message** allow a Common Graphics app to add one or more icons to the system tray, with custom mouse behavior.
7. **New function provides suitable default for stream argument to utility dialogs.** The stream returned by the newly exported function **selected-window-or-screen** is now the default value for the `owner` argument to **ask-user-for-new-or-existing-directory** (instead of the screen). The other utility dialog functions default to this value as well, which is essentially the same behavior as before. These functions include **ask-user-for-choice**, **ask-user-for-choice-from-list**, **ask-user-for-color**, **ask-user-for-directory**, **ask-user-for-existing-pathname**, **ask-user-for-font**, **ask-user-for-string**, **pop-up-message-dialog**, **pop-up-printer-setup-dialog**, **pop-up-string-dialog**, **pop-up-strings-dialog**, **pop-up-find-dialog**, **pop-up-replace-dialog**, **yes-no-or-cancel-list**.
8. **New parenthesis matching and reindenting operators; parenthesis matching now available on Common**

Graphics applications. The generic functions **parenthesis-matching-color** and **parenthesis-matching-style** are now available in Common Graphics applications (in earlier releases they worked in the IDE only). The new generic function **parentheses-matched** allows customizing behavior when matching occurs, for parentheses and double quotes. (You might, for example, display the matched parenthesis in the status bar if it is out of view. The default method does that in the IDE but does nothing in a Common Graphics application.) The macro **without-parenthesis-matching** provides further control over parenthesis matching.

The new generic functions **reindent-region** and **reindent-single-line** redisplay Lisp source text following Lisp indentation rules.

9. **justification is now a property of the button.** The **justification** property now applies to `button` widgets.
10. **Newly exported functions `windows:set-default-command-line-arguments` and `windows:set-application-icon` for specifying the command-line arguments and icon of an application.** The function **`windows:set-default-command-line-arguments`** allows you to set the command-line arguments of an executable (typically one you have created for an application). **`windows:set-application-icon`** allows you to specify the icon used for the application.
11. **New `unpress-automatically` property for `picture-buttons` and `multi-picture-buttons`.** The new `picture-button` and `multi-picture-button` property **`unpress-automatically`** will cause any button that was pressed interactively to be unpressed just after the widget's **`on-change`** function (if any) has run. This is clearer than the obscure technique of returning `nil` from the **`on-change`** function to "reject" a button press, causing the button press to be undone.
12. **New project property `ide:additional-build-lisp-image-arguments`.** The setf'able generic function **`additional-build-lisp-image-arguments`** allows specifying additional arguments to **`build-lisp-image`** in a project. This project property is supplied as a catch-all for any **`build-lisp-image`** arguments that are not covered by more specific project properties.
13. **New functions `do-click`, `do-keypress`, and `do-keypresses`.** The functions **`do-click`**, **`do-keypress`**, and **`do-keypresses`** programmatically emulate clicking a mouse button or pressing and/or releasing a key on the keyboard, or pressing all the keys associated with characters in a string. These functions may be useful for automated testing.
14. **`defproperties` forms will fill in missing `define-property` symbols.** A call to **`defproperties`** no longer requires that each property definition begin with the symbol **`define-property`**; it can simply be left out.
15. **`update-multi-picture-button` no longer needed and calls to it should be removed.** Use of the function **`update-multi-picture-button`** is deprecated as it is no longer needed. We recommend that you remove any leftover calls to this function in an application. Doing so will have no effect. (The update is done automatically using `:after` methods on `button-info` property setters.)
16. **New `no-clip` optional argument to `draw-string-in-box`.** The function **`draw-string-in-box`** has a new optional `no-clip` parameter for suppressing the usual clipping of the string at the specified box.
17. **New function `top-clipboard-value-of-type` allows access to most recent clipboard item of a particular kind.** The new function **`top-clipboard-value-of-type`** allows retrieving a clipboard value that may have been pushed off the clipboard by a value of another type.
18. **New `on-change` argument to `yes-no-or-cancel-list`.** The function **`yes-no-or-cancel-list`** has the new keyword parameter `on-change`, which is exactly like the `on-change` parameter of **`ask-user-for-choice-from-list`**. (If the value is a function, it is run when the user highlights a choice in the displayed dialog.)
19. **Increments of an `up-down-control` can vary over time as a mouse button is held down.** You can specify that an `up-down-control` increments by different amounts as the mouse button is held down. See **`increment`**, where the alternate value accepted for `up-down-controls` is discussed.
20. **New functions `string-search` and `string-replace`.** **`string-search`** and **`string-replace`** search text in `text-edit-panes` and `rich-edit-panes` for a specified string and return its location (**`string-search`**) or replace it with a different string (**`string-replace`**).
21. **New functions for custom `dde:convert-returned-dde-buffer` methods.** A custom **`convert-returned-dde-buffer`** method may now call either of the new functions **`string-from-dde-buffer`** or **`dword-list-from-dde-buffer`** to handle the common cases where a DDE server returns a null-terminated string or a vector of longwords. The DDE module has been separated from Common Graphics. See [Appendix D Details of separating DDE code from Common](#)

[Graphics.](#)

22. **Newly exported dde classes server-port and dde-port; open-server now has a server-port-class argument.** The symbol naming the `server-port` class has been exported. It and `client-port` are the instantiable dde port classes. `dde-port` is the superclass for the dde port classes. You do not instantiate a `server-port` instance; this is done by **open-server**. **open-server** now has a *server-port-class* keyword argument allowing you to specify the server-port class to open. It is often useful to subclass `client-port` and `server-port` to allow specialized **dde-message** methods.
23. **:default now a suitable value for cursor:** The value passed to (**setf cursor**) may now be the symbol `:default` to use whatever is the default mouse cursor for that object. This is also now the default value for windows.
24. **box-intersect now has an optional box-to-return argument.** The function **box-intersect** has a new optional argument called *box-to-return*. If a box object is specified as its value, that box is modified and returned as the result, thus avoiding consing a box structure on each call.
25. **Various box information functions that return positions now have an optional position-to-return argument.** The functions **box-top-left**, **box-top-right**, **box-bottom-left**, **box-bottom-right**, **box-center**, **box-left-center**, **box-top-center**, **box-right-center**, and **box-bottom-center** all return positions. Each now has a new optional argument called *position-to-return*. If a position object is specified as its value, it will be modified and returned, thus avoiding consing a new position object on every call.
26. **Documentation for pprint-plist-definers and pprint-plist-pairs-on-separate-lines rewritten.** See **pprint-plist-definers** and **pprint-plist-pairs-on-separate-lines**.
27. **Find In Files dialog now displays HTML files in a HTML browser.** The **Find In Files** dialog will now display files that have an HTML file type in an HTML browser program rather than in the IDE editor. HTML files are identified by their type, with the types *htm*, *html*, and *shtml* all identifying HTML files.
28. **IDE dialog that announces application has been built allows launching the application.** When the IDE's **File | Build Project Distribution** or **File | Build Project Exe** command has finished generating the standalone application or executable, the dialog that mentions that it is finished has a new button for starting up the application or executable.
29. **A bug fix with regard to bignum position and box coordinates.** The functions **position=** and **box=** always returned `nil` when any of the coordinates were bignums. This is fixed.
30. The default prompt that was documented for **get-screen-box** has now been implemented.
31. **New MCI operators mci-delete and mci-set-wave-options; new example of MCI recording.** **mci-delete**, a new generic function, allows deleting all or part of the recorded sound in an `mci-wave-audio` device. **mci-set-wave-options**, a new function, allows setting the audio quality and other options when recording a wave file. There is a new example on the **mci-record** page which uses the new operators.
32. **New function timer-process.** The new function **timer-process** returns the process in which the timer was most recently started and in which it is handling timer events.
33. **New function ellipse-start-and-end.** The new function **ellipse-start-and-end** may be used to find the coordinates of the endpoints of an arc drawn with **draw-ellipse-arc**.
34. **New generic functions select-adjacent-tab and select-recent-tab change the selected tab on a tab control.** The new generic functions **select-adjacent-tab** and **select-recent-tab** are useful for keyboard shortcuts that select nearby or recently-selected tabs of a `tab-control`.
35. **New operators object-locale and with-object-locale allow setting locale for individual controls and binding locales during execution of code.** The new generic function **object-locale** allows setting the locale of an individual window or widget, which may affect the character set that it uses. The new macro **with-object-locale** allows executing a body with `*locale*` bound to an object's individual locale.
36. **Menu item title may now be a pixmap.** The title of a `menu-item` may now be a pixmap in order to display an image rather than a string in the menu.
37. **New methods for determining background and foreground colors of outline-items.** The new overridable generic functions **outline-item-selected-background-color** and **outline-item-selected-foreground-color** allow customizing the colors that are used to draw the selected item of an outline control.

38. **New generic function to provide control over outline controls.** Increased control over outline widgets is provided by the new functions **select-outline-item**, **selected-outline-item**, and **focused-outline-item**. The function **outline-item-parent** may now be passed an `outline-item` for efficiency rather than the item's `value`. And the generic functions **first-visible-line**, **set-first-visible-line**, and **number-of-text-lines** now apply to outlines.
39. **New IDE introduction.** A new high-level interactive introduction to the IDE is available both on the **Help menu** and on the new **Startup Action** dialog.
40. **grid-widget documentation expanded.** The documentation of the `grid-widget` class has been rewritten and expanded.
41. **New filename keyword argument to pop-up-printer-job-dialog.** **pop-up-printer-job-dialog** now accepts a filename keyword argument which specifies the file to which a printer job should be sent (when you do not want the material actually printed). The argument is actually passed to **open-stream**.
42. **New function position-of-character-index returns the position of a particular character.** **position-of-character-index** returns the position given the index in a `text-edit-pane` or a `rich-edit-pane`; the related (and already defined) **character-index-at-position** gives the index given the position. Note that **character-index-at-position** returns `nil` when the position is not visible in the pane, as there is no way to determine the position in that case.
43. **New implementation of invoke-html-browser, invoke-html-browser-using-dde uses old implementation.** In earlier releases, **invoke-html-browser** used a DDE interface to invoke a running Internet Explorer or Netscape 4.x (if any). This is no longer done, and instead a technique that emulates keystrokes in browser is used, as was done already with Netscape 6.x and Mozilla, and with all four browsers by **invoke-private-html-browser**. This change was made for consistency now that the newer technique has proven itself, and to avoid a couple of problems with the DDE interface. If this (unexpectedly) leads to a backward incompatibility, please see **invoke-html-browser-using-dde**, which uses the older **invoke-html-browser** implementation.
44. **The macro without-on-change now works with multiple widget arguments.** The **on-change** handler for each specified widget will be temporarily set to `nil` while the body of the **without-on-change** macro executes. This change is backward compatible. (Previously, only one widget could be specified.)
45. **New setf'able function test-edit-margins.** The **setf** of new function **text-edit-margins** can be used to create some space between the text of a text-editing control and the control's left and right borders.
46. **Find in Files dialog now has Case-Sensitive checkbox.** The **Find in Files** dialog has a new check-box for case-sensitive searches. When checked, the search string "window" will not find the string "Window". When unchecked, it will. The initial value is unchecked. Previously all searches had been non-case-sensitive (corresponding to unchecked).
47. **Changes to the trace dialog.** The **Trace Dialog** has a new widget that shows the effective method of any traced generic function call, providing access to each individual method of the effective method. (This information is also now printed by the base Lisp trace facility.) The **scroll-while-tracing** configuration option may now be toggled directly on the **Trace Dialog**.
48. **New centered property, :centered argument to make-window, and center-window generic function.** The **centered** property and the `:centered` argument to **make-window** allow specifying that a window be centered over its **parent** or **owner** on creation, while **center-window** causes an existing window to be centered over its parent or owner.
49. **Tools for removing methods using the class browser.** Methods may now be removed from the Lisp environment in the **Class Browser** dialog by invoking the **Edit | Delete** command in the Methods tab or in the **All Methods** tab. (Removing methods was already possible in the **Definitions** dialog.)
50. **use-private-html-browser now defaults to true.** The default value of the IDE's **use-private-html-browser** configuration option is now `t` rather than `nil`, so that the IDE's help facility will not reuse a browser that you were using for something else. (The symbol naming that function is now in the `ide` package, moved from the `cg` package. See [Appendix B Details of Common Graphics/IDE package reorganization](#) for more details on the package changes.)
51. **The minimize-button and the maximize-button on a Window can now be enabled and disabled at runtime.** In previous distributions, the buttons could be enabled only at window-creation time. See **minimize-button** and

maximize-button.

52. **mouse-*-down and mouse-*-up changes.** **mouse-left-down** etc. are called more consistently for widgets. The extendable generic functions **mouse-left-down**, **mouse-middle-down**, **mouse-right-down** and the corresponding "up" functions had been called on the `dialog-item` object for an `os-widget` but on the associated `widget-window` object instead for a `lisp-widget` (or on both for an `os-widget` on which **subclass-widget** had been called). Now these functions are called on both the `dialog-item` and the `widget-window` for any type of widget, so an application could add methods for either type of object. Actually, the "up" functions had not been called at all for `os-widgets`.
53. **Additional optional arguments for extract-icon-from-file and draw-icon.** The functions **extract-icon-from-file** and **draw-icon** have additional arguments that allow extracting and drawing a small (16 by 16) icon at its defined size rather than stretching it to standard size (32 by 32).
54. **New hotspot functionality.** Hotspots have been enhanced so that the active region may be a polygon, a circle, a line, or (as before) an orthogonal rectangle. See the description of the `hotspot` class.

The hotspot accessor that returns or sets the active region of the hotspot has been renamed from **box** to **hotspot-region**, to reflect the fact that the active region of a hotspot no longer need be a simple box. For compatibility, the now-deprecated **box** and (**setf box**) methods for hotspots still exist, and the `:box` initarg still works. The new generic function **region-box** returns the smallest rectangle containing a hotspot region. This rectangle is suitable as the area to invalidate when a hotspot area should be residplayed.

The **cursor** property now applies to hotspots, so (**setf cursor**) or the `:cursor` initarg may be used to associate a mouse cursor with a hotspot, which removes the need for an application to change the hotspot dynamically via **mouse-in** and **mouse-out** methods on the hotspot when a different cursor is desired over the hotspot.

The generic function **highlight-hotspot** has an added keyword parameter named *off* that indicates whether the highlighting is being turned on or off. Custom methods that do not draw in `po-xor` mode can use this argument to know whether to draw or erase the hotspot highlighting. Any existing custom methods defined in applications must add the new parameter.

Finally, the documentation for `hotspot` and various hotspot related functions, including **invalidate**, has been expanded and the bug has been fixed where a hotspot could be left highlighted if you move the mouse cursor out of the hotspot and also out of its parent window without any **mouse-moved** messages happening on the parent window in the meantime.

55. **New Granularity and Max Samples widgets on Runtime Analyzer Control Dialog.** The Granularity widget sets the value of `prof:*granularity*`. The Max Samples widget sets the value of `prof:*maxsamples*`. See the description of the **Runtime Analyzer Control** dialog for details.
56. **New generic function highlight-tab highlights a tab on a tab-control.** See **highlight-tab** and `tab-control`.
57. **New IDE option highlight-selected-editor-tab makes current editor tab more obvious.** See **highlight-selected-editor-tab**.
58. **Additional widgets now have the on-double-click property.** **on-double-click** is now available for `group-box`, `tab-control`, all instantiable subclasses of `text-widget` (`static-text`, `editable-text`, `multi-line-editable-text`, `lisp-text`, `multi-line-lisp-text`, and `rich-edit`), and `combo-box`. Note for a `combo-box`, an **on-double-click** function is useful only when the **typable** property of the `combo-box` is true, since otherwise the two clicks would simply drop and undrop the drop-down list.
59. **The on-mouse-in and on-mouse-out properties now exist for header-control.** See `header-control`.
60. **New IDE option directory-dialog-avoids-network.** The **directory-dialog-avoids-network** option could be enabled if you notice a long delay before a dialog appears in the IDE to ask you for a new directory. This option would skip the search for directories on other machines in the network, which is probably the reason for any significant delay that might occur.
61. **New options to colorize source code in editor buffers.** See **colorize-source-code** and the new **Editor Color** tab on the **Options** dialog in the IDE. Colorizing code allows immediate identification of comments, strings, and other types of objects. There are eight different things (like comments and strings) that can be colorized and each has (originally) a different color. Other colors can be used instead. See **colorize-source-code** for a list of things colored and the

accessors to the colors used.

62. **New generic function `private-html-browser-handle` allows ensuring specific browser instances are used when desired.** `private-html-browser-handle` identifies the particular instance of the default HTML browser program that is currently being used by the function `invoke-private-html-browser` to show HTML pages from lisp. This value can be stored when Lisp exits and then reused when it starts again, if the browser is available.
 63. **New generic function `retain-scrollbars`.** `retain-scrollbars` allows a program to control whether a window will always have scrollbars even when there is nothing to scroll (that is, all content is already visible). Because scrollbars take up part of the interior of a window, one might wish to retain them so that the interior size is always the same, regardless of whether scrolling is necessary.
 64. **New function `generate-mask`.** `generate-mask` creates a `mask` for a pixmap.
 65. **Fonts can now have an angle specified; text will be drawn at that angle.** Both `make-font` and `make-font-ex` have a new angle argument which allows specifying the angle at which text written in the font will be drawn. The default value (`nil`, equivalent to zero) specifies that text will be drawn horizontally, as usual. A non-negative integer value specifies the angle which the text will be rotated counterclockwise about the upper-left corner of the string when horizontal (thus 90 is vertical, first letter lowest, 180 is upside down and 270 is vertical, first letter highest). The new function `font-angle` returns the angle of a font. If a font has a non-zero angle, `draw-string-in-box` should not be used to draw a string in the font as the rotating may move the text out of the box; standard Common Lisp output functions like `format` and `princ` should be used instead.
 66. **New property `double-buffered` and new macro `with-double-buffering` can prevent flashing when modifying a window.** The flashing typically occurs during scrolling, resizing, or animation. See `double-buffered` and `with-double-buffering`.
 67. **New subkey argument to `add-application-window` and related functions.** `add-application-window`, `remove-application-window`, and `find-application-window` all have a `subkey` optional argument which further specifies the window being added or removed or sought. (This argument defaults to the current process, so different processes can use the same key (typically the name) of windows but still have them distinguished.) `add-application-window` also has an `always` optional argument which can force the replacement of an existing cached window with the new one. The descriptions of all three functions have been updated.
 68. **New property `override-menu-bars` allows overriding menu shortcuts.** When a keyboard command is defined in a `comtab` (command table) and also as a menu shortcut, the system has to determine what to execute when the command is entered. The `override-menu-bars` allows determining which should have precedence. When `nil` (the default, preserving the old behavior), the menu command has precedence. When non-`nil`, the `comtab` command. In the IDE, this property is true, so `comtab` command override menu shortcuts. See `*text-edit-comtab*`.
 69. **Any icon file may now be used for an application generated on NT.** When generating a standalone application from an IDE project, any `.ico` file will now work for the application icon as long as you are generating the exe on an NT branch of Windows (which includes win2000 and XP). This is done internally by calling the new function `win:set-exe-icons-nt`, which you could also call directly. Formerly (with `win:set-exe-icons`) the `.ico` file needed to contain exactly one or two images, which had to be 32x32x4 and/or 16x16x4. This restriction still applies to the win98 branch, and the project system will reject other icons on those platforms only. The NT restriction does not apply to end users who run the generated application.
 70. **New generic function `internally-loaded-files` allows more files to be searched to determine necessary modules for a project.** There is a **Find Required CG Modules** button on **Project Manager** that causes the system to examine project files to see which CG modules will be needed in the application. You may specify additional files to be searched using the `internally-loaded-files` property. Any files appearing in the list which is the value of this property will be examined along with other project files. (You may have arranged files to be included in a project without making them actual project files. In that case, adding them to the `internally-loaded-files` list ensures they will be examined.)
-

8.3 IDE release notes

The bullets in this section discuss general changes. The subsections [Section 8.3.1 The console window and tray icon in applications](#) and [Section 8.3.2 Opening projects from releases prior to 7.0](#) discuss more involved topics. See also [Appendix B Details of Common Graphics/IDE package reorganization](#).

- **New dialog reminds you to check for new patches.** The new **Check for New Patches** dialog is displayed periodically to remind users to check for new Allegro CL and Common Graphics/IDE patches. You can control how often this dialog appears either by specifying a value of the dialog itself or by setting (with **setf**) the value returned by **patch-reminder-interval**.
- **Most system messages now go to IDE listener rather than the console.** The IDE GUI process now binds the standardized I/O customization variables such as `*standard-output*` to the currently selected IDE listener rather than the console. Any information that the IDE had printed to the console, such as the generate-application call that is made by **File | Build Project Distribution**, will now appear in an IDE listener instead.
- **Project name and directory displayed in Project Manager.** The **Project Manager** dialog will now display the name of the project in its title-bar, as in "Project Manager - Employee-Directory". And the project's default directory will be displayed just above the list of modules on the **General** tab.
- **Special character insertion command on Edit menu.** The IDE Editor's Alt-Q gesture to insert any (typically untypable) Latin1 character has been moved to the **Edit menu** (as Control-Q) so that it may be used in any text-editing control, such as in the modal **Search | Find** dialog or the **Search | Find in Files** dialog.
- **Alt-Q no longer works to install a special character in an editor, control-shift-Q instead of control-Q evaluates the clipboard.** Related to the previous entry, in earlier releases the editor comtab had an entry that made the gesture Alt-Q insert a special character. As noted, it has been replaced by a menu command with shortcut Control-Q. Alt-Q is no longer defined as a gesture. Also, control-Q was the shortcut for the **Tools | Evaluate Clipboard** command. That shortcut is now control-shift-Q.

8.3.1 The console window and tray icon in applications

Turning off the **Enable Debugging of Runtime Errors** (previously called **Allow Runtime Debugging**) option for a project (see the **Build** tab of the **Project Manager Dialog**) had not actually removed the system tray icon for the Allegro console window in the standalone app as claimed in earlier releases. This is now fixed by internally passing the `+c` command line argument to avoid creating the console window at all in the standalone app.

If a project uses a custom icon for its standalone application, then that icon will also be used for the console window of the generated application and for the console's icon in the system tray (though these items typically are not present in a delivered application). Previously the Franz bust icon was used for the console.

The `+t` command line argument is no longer used in a special way by a Common Graphics app. A project had defaulted the `+t` command line argument for the console's title in its standalone application to "Initializing", and then Common Graphics had replaced that title with "Console" after the app finished starting up, to avoid a problem where the console title had appeared momentarily in the taskbar button for the app as it started up. This was actually not necessary when using the `+cx` argument to keep the console window hidden completely, or when using `+c` to avoid creating the console window at all, as is now done for projects, and so this hack has been removed.

8.3.2 Opening projects from releases prior to 7.0

When you open a project in the IDE for further development, and the project was most recently saved in a pre-7.0 release of the IDE, then a few project options will not be preserved, though these can easily be updated if needed on the **Project Manager** dialog. These options are:

1. The set of optional CG modules that are included in the standalone application that is generated from the project. The

current release has many additional modules, which may allow you to make a smaller standalone CG app than before, but the old set of modules is not compatible with the new set, and so the modules must be respecified. The good news is that the new **Include** tab of the **Project Manager** dialog has a new button that generally will automatically determine which modules are needed by the current project.

2. The option for enabling debugging at run time has been renamed, and will initially be turned off when opening a project from an earlier release. On the **Build** tab of the **Project Manager** dialog, the check-box that had been labeled **Display Allegro Icon in System Tray** is now at the top of the widgets on that page and is labeled **Enable Debugging of Runtime Errors**. This was done to clarify its meaning and importance and to group it with the option for debugging build errors. The **build-flags** member for this option has been renamed from `:suppress-systray-icon` to `:allow-runtime-debug`, and has the opposite meaning as before. The system tray icon will still appear when this option is on, but that is no longer the significant feature of this option, since the console will now automatically appear for debugging a runtime error when this option is on, and so the system tray icon is no longer needed for this purpose.
3. Similarly, the **Build** tab check-box that had been labeled **Allow Debug of Build Error** is now labeled **Enable Debugging of Build Errors (console will not exit)**. The corresponding **build-flags** member has been renamed from `:allow-debug` to `:allow-build-debug`. When this option is selected, it will now pass the *build-debug* keyword argument to **generate-application** as `:interactive` rather than as `t`. This allows actually debugging a build error that has occurred, rather than simply seeing the error message in the leftover console window after the lisp that builds the standalone image has exited. This is more useful, but also entails leaving the build console up on every build, even if no build error occurred. To avoid confusion when the console does not exit, this option will no longer be turned on initially when a project from an earlier release is opened, and the option is no longer on by default for a new project. The check-box widget also mentions that the console will not exit, as a reminder.

Also, the new **include-flags** value `:debugger` determines whether the debugger is included in the standalone app, and there is a check-box for it on the new **Include** tab of the **Project Manager**. It is included by default. In earlier releases the project system always included the debugger. There are also individual check-boxes for including the compiler and the top-level, which had been mixed in with the list of CG modules. And a new check-box for saving local name info.

8.4 New behavior when starting the IDE or creating a new project or form

There is a new **Startup Action** dialog that appears by default when the IDE starts up. It asks whether to open a recently-used project, open some other project, create a new project, or proceed without a current project. (This dialog may be disabled with the check-box near the bottom of the dialog, or on the **Options dialog**.)

It is now possible to run the IDE without a current project at all, which is what happens if the new startup dialog is disabled or canceled. A project may still be opened or created any time later in the usual way (with, for example the **File | Open Project** menu command). There is also a new toolbar button on the **Project Manager** dialog to close the current project without opening another one. You can tell whether there is a current project by checking whether a project name appears at the left end of the main IDE title-bar.

When creating a new project, the directory-choosing dialog first appears asking for a default directory for the project's files. A project's files do not have to be placed into this directory, but having a default directory helps the IDE later to default the file-selection dialog to a directory that is appropriate for the project.

Additionally, the first time a project is created there will be an earlier invocation of the directory dialog asking for a default parent directory for all new IDE projects. This default parent directory is simply used as the initially-selected parent directory whenever a new project asks for a default directory for that particular project. This dialog will not reappear as long as the *prefs.cl* file is created and retained as usual, though the value could still be changed by setting the new **project-parent-directory** configuration option.

When creating a new form, the **New Form** dialog that asks for the window class to instantiate now has two additional widgets. One widget asks for an optional new subclass of the selected existing class, and the other asks for a name for the

form. The name is used for the default filename for the form module, and so naming the form at creation time helps to avoid saving the form to a randomly-named file. This name is also used for the **finder-function** and **maker-function** that are generated for the form; if the dialog name is "foo", then the **finder-function** will be named **foo** and the **maker-function** will be named **make-foo**.

When a new form is created, an editor buffer is now always created immediately for the `.cl` file for user-written code for the form. Previously, this buffer was created lazily only when the first event-handler skeleton function was generated for a widget on the form (by clicking on an event-handler button in the inspector). This may have been confusing if you had decided to write some code for the form before this time. If you specify a new subclass for the form on the **New Form** dialog, then a **defclass** expression for the new form class is written into this buffer. Some new comments are also added to the top of this buffer to explain what it is for.

The new configuration option **new-project-create-form** will automatically create a form module when creating a new project; this replaces the no-longer-used option **new-project-show-form**, which still exists for compatibility but now does nothing; this option is no longer on by default. In earlier releases, a form window was always created when a new project was created, but the dialog asking for the window class to instantiate was not shown, and so the "dialog" class was always used. Now the dialog asking for the window class to instantiate will always appear when a new form is created.

The **new-project-show-project-manager** option is now true by default, since there is an option to not create or open a project at all, in which case the project manager will never appear.

The new configuration option **open-project-show-files-in-editor** will automatically show all of the source code files of a project in the IDE editor when the project is opened. The new option **close-project-close-editor-buffers** will close any unmodified editor buffers from a project when the project is closed. Those options are disabled by default, but may be enabled on the **Project** tab of the **Options dialog** (displayed with **Tools | Options**). The new option **open-project-show-project-manager** will show the **Project Manager** dialog whenever a project is opened. That option is enabled by default.

When a project is opened, the Inspector is now shown if and only if any of the project's forms are visible, in which case the first such form is inspected. Previously the Inspector was always shown when opening a project. When creating a new project, the Inspector is shown only if the **new-project-create-form** option is enabled; creating a new form always shows the inspector in order to inspect that form.

When the IDE is started up with an initial project to open, such as by clicking on a `.lpr` file in the Windows File Explorer, the IDE will no longer exit if an error occurs while initially compiling or loading the project. This also applies to the new functionality for selecting a project to open at startup time.

When changing the name of the current project on the Options tab of the Project Manager dialog, and the `.lpr` project definition file has already been written, a modal dialog will ask if you also want to rename the `.lpr` file to match the new project name. These names are the same initially, and so it may reduce confusion to keep them in sync.

When typing a package name into the package widget on the Options tab of the Project Manager dialog, the package no longer needs to exist beforehand. If a new package name is entered, the package will now be created automatically, and a minimal `defpackage` form for the package will be written to the generated `.lpr` project definition file for the project (as before) to ensure that the package is created when the project is opened or loaded into a later lisp session or standalone application. Also, the drop-down list of package choices for this widget no longer includes packages that exist when the IDE is starting up (except for `cg-user`); this is done to exclude built-in packages that aren't appropriate for user code.

When a source code file is added to the current project either with the Project Manager's "Add File" button or the editor's "Add File to Project" command on its right-button menu, and the file is not currently in the project's default directory (or some descendent directory of it), then a yes-or-no dialog will ask whether you want to move the file into the project's default directory. The file will be moved there automatically if you answer yes.

When saving a new IDE editor buffer for the first time, a yes-or-no dialog will first ask whether you want to add the new file to the current project. (This dialog will NOT appear if there is no current project, or for a form's editor buffer since that is already in the project.) If you answer yes, then the Save File dialog will default to the project's default directory, and the file will be added to the project after it is saved. If you answer no, then the Save File dialog will default as it otherwise

would, which is generally to the directory in which a file was last selected in this dialog, and the file will not be added to the current project.

When clicking on the Components toolbar, then the frontmost visible form (if any) is brought all the way to the front. If there are no visible forms, then a dialog will tell you to first create or open a project.

9.0 Release Notes for AllegroStore

The AllegroStore manual is available online in PDF format, [allegrostore.pdf](#). A printed copy is available.

9.1 Non-backward-compatible changes in AllegroStore

AllegroStore 2.8 requires a new version of ObjectStore (as described in [Installation instructions and information for AllegroStore users in installation.htm](#)).

9.2 Other changes in AllegroStore

The version of AllegroStore shipped with Allegro CL 7.0 is 2.8. The underlying ObjectStore database is "ObjectStore Release 6.1 Service Pack 1".

All 32-bit platforms should now be fully interoperable ("homogeneous") e.g. they can simultaneously share a database on a single server. There is no homogeneity with the HPUX 64-bit platforms because 32-bit and 64-bit platforms have different data representations. Databases can only be transferred between 32- and 64-bit platforms by using `asdump` and `asrestore`. Contact Franz Inc. support at support@franz.com if you need to do this.

10.0 Release notes for SOAP

There were various releases of the SOAP interface in Allegro CL 6.2. This section lists the changes from the two most recent SOAP updates.

Update in late summer, 2004

New features

- WSDL files can now be generated from the Lisp definition of a service.
- The namespace configuration in a WSDL file can now be examined and a suitable namespace-to-package mapping chosen.

Enhancements and corrections

- Can now parse and generate interfaces for 149 of 157 RPC definitions on `xmethods.com`.
- `soap-message-client` can now specify a proxy.

Miscellaneous

- New `:start` keyword argument to `soap-message-client`.
- New `:name`, `:service-name`, `:url`, `:port-name` keyword arguments to `soap-message-server`.

- New accessors for `soap-connector` class: **soap-port-name**, **soap-binding-name**, **soap-service-name**.
- New functions: **enable-soap-server**, **disable-soap-server**, **start-soap-server**, **stop-soap-server**, **wSDL-service-names**, **encode-wSDL-file**, and **decode-wSDL-namespaces**.
- **make-client-interface** and **make-server-interface** have new keyword arguments `:port`, `:if-missing-package`.

Update with 7.0 release

New features

1. Generate a WSDL file from the Lisp definition of a service.
2. Examine the namespace configuration in a WSDL file and choose a namespace-to-package mapping.

Enhancements and corrections

1. Extensive revisions of WSDL parsing and generation.
2. We can now parse and generate interfaces for 357 of 370 definitions on **xmethods.com** (both rpc and document styles).
3. Summary information is in generated client or server file.
4. **soap-message-client** can now specify a proxy.

Changed behavior

1. Revised the namespace-to-package mapping concept with new syntax and new operators (**define-namespace** and **define-namespace-map**).
2. **make-server-interface**, **make-client-interface**: the *eval* argument is ignored.
3. Namespace variables eliminated, replaced by namespace mapping.

Details

1. **soap-message-client**: new keyword arguments: *start trim-whitespace soap-debug base-dns*.
2. New accessors for `soap-connector` class: **soap-port-name**, **soap-binding-name**, **soap-service-name**.
3. New operators: **define-namespace** (to make a global namespace definition), **define-namespace-map** (to make a named, global namespace-to-package mapping).
4. New variables: **soap-client-debug** and **soap-server-debug**.
5. **make-client-interface** and **make-server-interface**: new keyword arguments *port* and *if-missing-package*.

11.0 Release notes for jLinker

The jLinker java-Lisp interface tool has been made more efficient. The most important changes are these:

- **New Java class `LispCall`**: this class (see the [description here](#)) along with **LispConnector** provide all the API that a Java program needs to call Lisp or to allow Lisp to call Java. Both **LispCall** and **LispConnector** are documented using JavaDoc (the links are to the links to JavaDoc in [jlinker.htm](#)).
- New JavaDoc documentation for some Java classes: **LispCall** and **LispConnector** are both documented using JavaDoc. The index to available JavaDoc documentation is linked to [here](#).
- **Function `javatools.jlinker:query-handler` renamed `javatools.jlinker:jquery-handler`**. The functionality is the same. The symbol `javatools.jlinker:query-handler` is no longer exported. Use **`javatools.jlinker:jquery-handler`** instead.

jLinker documentation is in [jlinker.htm](#).

12.0 Release notes for AllegroServe

AllegroServe is continually updated. A snapshot of the documentation is available at [aserve/aserve.html](#) but later versions may be available (the latest version is always available at <http://opensource.franz.com/aserve/>, scroll down for links to the documentation). Recent changes include:

- support non-standard http headers.
 - Webactions a web application framework (see [Section 6.1.1 Features added to Allegro CL 6.2 after the initial release of Allegro CL 6.2](#)).
 - Modification to handling of data generated by a cgi script: it is now returned to http client immediately rather than being buffered.
-

13.0 Release notes for IMAP and SMTP, XMLutils

IMAP Release Notes

New Imap macros and functions. **with-imap-connection** and **with-pop-connection** open a connection to execute body forms and then close the connection. **with-fetch-letter-sequence** reads a pop interface letter in parts. **fetch-letter-sequence** and **end-of-letter-p** are generic functions used by **with-fetch-letter-sequence**. Also new is **reset-mailbox** which removes any **delete** or **has been read** markers from letters in a pop mailbox.

New functionality for smtp mail servers that require authentication. Authentication is provided by providing a login name and a password. **send-letter** has new keyword arguments *login* and *password*. The new function **send-smtp-auth** is like the existing **send-smtp** except it has two additional required arguments *login* and *password*.

The authentication support is new. It works in our tests and with our mail servers, but, for whatever reason, may not work for everyone. If you experience problems with authentication, please send us a bug report.

XMLutils Release Notes

All changes since 6.2 have been released as patches. In **phtml** (see [phtml.htm](#)):

- New `:eliminate-blank-strings` argument to **parse-html**
- **parse-html** close tag closes consecutive identical open tags
- New `:parse-entities` arg to **parse-html**

In **pxml** (see [pxml.htm](#)):

- improved namespace handling
 - improved handling of character encodings
 - new **pxml-version** function
 - allow unparsable uri as namespace marker
 - improved performance
 - more correct parsing of DTDs
-

14.0 Release notes for The Emacs/Lisp interface

As said in [Section 6.1.1 Features added to Allegro CL 6.2 after the initial release of Allegro CL 6.2](#), there was a major upgrade to the Emacs-Lisp Interface released as a 6.2 patch. See the section **Significant changes in the interface** in [eli.htm](#) for more information.

In this section we list some minor changes and bug fixes.

- Non-ASCII characters in the communication between Emacs/MULE and Lisp are now properly handled.
- Some errors which happened when Lisp and Emacs communicated filenames have been fixed.
- symbol completion now works when `excl: *print-nickname*` non-`nil`. Also symbol completion of keywords, and other obscure completion problems have been fixed.
- Change emacs-lisp rendezvous on windows to be more like UNIX. This has two important/significant benefits to Windows users: (1) multiple instances of emacs/lisp pairings are now possible; and (2) by specifying a remote host name to **fi:common-lisp**, you can run a Lisp on a UNIX/Linux machine from an Emacs running on Windows. This requires an `rsh.exe`, but NT/2k/XP has a native version. Cygwin can be used if `rsh.exe` is not already available.
- The correct file position and name now appear in the popup buffer for incremental compiler warnings.
- On Windows, SHELL is unset before running Lisp, since the version used by Emacs on NT/2k does not work well with **run-shell-command**.
- There is a new variable `fi:user-env-vars`. It is an alist containing environment variable/value pairs to add to the environment of the Lisp started with **fi:common-lisp**.
- The pretty printer will notice resizing of an Emacs frame and act appropriately.
- Finding character positions in buffers in binary mode now works properly.
- Multiple in-package's in a file when doing evaluations of forms and definitions are handled properly.
- Indentation of forms when element after opening paren is `#+/ #-` pair are now handled like this:

```
(#-blah no-blah
  #+blah yes-blah
  (blah blah blah)
  t)
```

instead of like this:

```
(#-blah no-blah
      #+blah yes-blah
      (blah blah blah)
      t)
```

15.0 Documentation modifications in Allegro CL 7.0

The format of the documentation in 7.0 is similar to that of 6.2.

1. The documentation of **standalone-application** has been expanded.

16.0 Availability of CLX for Allegro CL

CLX (Common Lisp X) provides an interface between Common Lisp and the X window system. All versions of Allegro CL include a compiled version of CLX with the distribution. The *fasl* file is `code/clx.fasl`, loaded by evaluating `(require`

: `clx`). The Allegro CL products CLIM and Allegro Composer use CLX. Users wanting low-level access to an X server in Lisp may also want to use CLX. CLX is not supported by Franz Inc.

The sources to CLX are supplied with the regular Allegro CL distributions in the *contrib/clx/* directory. Note that during installation, you are asked whether you wish to install the *contrib/* directory and the default is not to install it. The *contrib/* directory is not included in the Trial distribution, but Trial users can download the CLX sources from the Franz Inc. website as described next.

The sources to CLX are also available on the Franz Inc. web site (www.franz.com), at the location <ftp://ftp.franz.com/pub/clx/>.

17.0 Release notes for Orblink

`wchar` and `wstring` types are now supported in Orblink. `wstring` maps to Lisp string; `wchar` maps to Lisp character. It is an error, however, to use `wstring/wchar` in an 8-bit character Lisp.

The POA documentation is now documented with the rest of Orblink.

Appendix A: Tightening of ANSI Conformance in Allegro CL

Working with Paul Dietz, a user of Allegro CL who has been developing an extensive test suite for ANSI Common Lisp, we have corrected a large number of non-conformances uncovered in Allegro CL. While Franz Inc. has always corrected non-conformances as they have been discovered by users or internally (the fix being either supplied at once as a patch or in a subsequent release), Paul Dietz's project (a comprehensive test suite for ANSI CL) has resulted in an unusually large number of changes.

Prior to Paul's test, newly uncovered non-conformances were identified by problems in running code: a user or a Franz developer would write what he or she believed to be conforming code but found that the code did not work as expected. Discussions with the relevant Franz developer would on occasion reveal that Allegro CL had a bug or that Franz had misinterpreted the specification.

Allegro CL 7.0 contains fixes for many of the non-conformances. In the large majority of cases, the fix will not affect existing code. In a few cases, code which previously worked will now fail. We list those cases next. Then we list the (few) cases where the bug is not fixed in the 7.0 release. Finally, we list the other fixes.

Appendix A.1 Conformance fixes which might break existing code

1. **do-style iterators now fully wrapped in an implicit block named nil:** `do`-style iterators (**`do`**, **`do*`**, **`dotimes`**, **`dolist`**, **`do-all-symbols`**, **`do-external-symbols`**, **`do-symbols`**) should be surrounded by a block named `nil`. Allegro CL prior to release 7.0, the null block was inside the initialization forms of the `do`-style form. In 7.0, it is outside. This means that calls to **`return`** within the initialization forms of a `do`-style form which previously returned from an outer null block will now return from the `do`-style form. The difference can be seen in the following example:

```
;; 6.2 behavior
(block nil (dotimes (i (return 11) 5)) 0) RETURNS 11
```

```
;; 7.0 behavior
(block nil (dotimes (i (return 11) 5)) 0) RETURNS 0
```

2. **Trying to print a specialized array when `*print-readably*` is true signals an error.** In earlier releases, a specialized array would always print in `#(format`, even if `*print-readably*` were true:

```
cl-user(1): (make-array 10 :element-type 'single-float :initial-element 1.0)
#(1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0)
cl-user(2): (let ((*print-readably* t)) (print *))

#(1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0)
#(1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0)
cl-user(3):
```

This was a nonconformance, because when the array is read back in, the resultant array will not be similar to the printed array (it will be an unspecialized array with element-type `t`).

Starting in release 7.0, an error of type `print-not-readable` will be signaled in this case:

```
cl-user(2): (let ((*print-readably* t)) (print *))

Error: Unable to print #(1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0)
      readably and *print-readably* is true.
      [condition type: print-not-readable]
```

A program which counts on the nonconformant behavior will break in 7.0. It may be possible to work around the compatibility issue by binding `*print-readably*` to `nil` before printing the array. This will cause the array to be printed in `#(format`, which will then allow it to be read in as before, though the resultant array will, as before, not be of the same element-type as the original.

3. **`fboundp` now errors when passed an invalid argument, `function-name-p` test whether argument is a valid name.** `fboundp` was out of spec in earlier releases in that it returned `nil` rather than signaling an error when passed an argument which was not a valid function name or specification. (Thus, e.g., `(fboundp 3)` returned `nil` rather than signaling an error.) `fboundp` now signals an error when passed an invalid argument. The new function **`function-name-p`** returns true if its argument is a valid function spec (and thus a suitable argument to **`fboundp`**) and returns `nil` otherwise. `(if (function-name-p spec) (fboundp spec))` thus behaves in release 7.0 as `(fboundp spec)` did in earlier releases.

Note that Allegro CL allows lists other than `(setf symbol)` to name functions. See **`def-function-spec-handler`**. **`function-name-p`** returns true when passed a valid spec defined by **`def-function-spec-handler`**. **`fboundp`** does not signal an error when passed such a function spec. (This makes **`fboundp`** arguably out of conformance.)

4. **Second argument to `documentation` is now required.** In earlier releases, the second (*doc-type*) argument to **`documentation`** was optional. It is now required, as called for by the ANSI spec.
5. A negative integer `:count` argument given to the sequence functions **`remove`**, **`remove-if`**, **`remove-if-not`**, **`delete`**, **`delete-if`**, **`delete-if-not`**, **`substitute`**, **`substitute-if`**, **`substitute-if-not`**, **`nsubstitute`**, **`nsubstitute-if`**, and **`nsubstitute-if-not`** was in earlier release (incorrectly) treated as `nil` (meaning act on all instances). Such values are now (correctly) treated as equivalent to specifying 0 (meaning act on no values).
6. **Going back from `pathname` / now signals an error.** In earlier releases, `(make-pathname :directory '(:absolute :back))` returned `#p"/` (i.e. the same as `(make-pathname :directory '(:absolute))`), so `:back` from `/` was ignored). Following from that, `(merge-pathnames "../a/" "/")` returned `#p"/a/` (i.e. back from `/` was `/`, and then the subdirectory `a/` was merged in). This behavior was incorrect. Allegro CL now signals a file-error in this (and similar) cases.
7. **Comparison of a rational and a float now coerces the float to a rational rather than the rational to a float.** The correct way according to the standard is coercing floats to rationals before comparing, and Allegro CL is now in compliance. Note that for each float, there are many, many rationals equal to it when floated. So floating the rational can often produce an identical number (which are, therefore, `=`, `<=`, and `>=` and not `<`, `>`, or `/=`) while rationalizing the float produces different numbers (which are not `=`, are `/=` and are either `<` and `<=` or `>` and `>=`). This means that

certain comparisons will now return different results. For example:

```
(setq num 121694457621910027047283135086593)
(setq den 121694457621910022543683507716096)
(= (/ num den) 1.0d0) -> nil
```

In earlier releases, that comparison returned true as the difference of the ratio (`/ num den`) from 1 was smaller than double-float-epsilon so `(coerce (/ num den) 1.0d0)` returned 1.0d0. Also any numerator value between **den** and **num** will also float to 1.0d0, and so also equaled 1.0d0 in pre-7.0 and does not (with the exception of the numerator equaling **den**) now.

While it is unlikely that any code depended on the results of comparisons of such numbers, the older `(coerce [rational] 'float)` behavior can be obtained by changing `(op fl rat)` to `(op (- fl rat) 0)` (and vice versa), where **op** is one of the numeric comparisons `=`, `/=`, `<`, `>`, `<=`, or `>=`.

- Named loop not surrounded by a block named nil.** Pre-7.0, **loop** placed an implicit null block around itself, but the specification is explicit in saying that such an implicit block is only added when the loop does not have the named option. Thus forms like this

```
(loop named foo ... ... (return res))
```

must be changed to

```
(loop named foo ... ... (return-from foo res))
```

- Correction of defstruct constructor bugs involving defaults.** Define two structures:

```
(defstruct (foo (:constructor make-foo-boa (&key bar bas)))
  (bar 10) (bas 'bye))
(defstruct (bar (:constructor make-bar-boa (&key (:foo bas)))) (bas 10))
```

In pre-7.0, we had the following results:

```
(make-foo-boa) -> #S(foo :bar nil :bas nil)
(make-bar-boa :foo 20) -> #S(bar :bas 10)
(make-bar-boa) -> #S(bar :bas 10)
```

In 7.0, we have:

```
(make-foo-boa) -> #S(foo :bar 10 :bas bye)
(make-bar-boa :foo 20) -> #S(bar :bas 20)
(make-bar-boa) -> #S(bar :bas nil)
```

Appendix A.2 Other conformance fixes

- New variable `excl:*strict-probe-file*` controls behavior of `probe-file` when file is inaccessible.** When given a pathname in a directory that is not readable by the user running the Lisp process (because, for example, the user does not have read permission in that directory), **probe-file** should signal an error. Allegro CL has returned `nil` rather than signaling an error. The variable `*strict-probe-file*` now controls the behavior of **probe-file** in this case. When `nil`, **probe-file** returns `nil`, and when non-`nil`, **probe-file** signals an error (and reports access problems). The initial value is `nil`.

Appendix B: Details of Common Graphics/IDE package reorganization

The Common Graphics and Integrated Development Environment (IDE) code has undergone an extensive reorganization for release 7.0. Among the reasons for the changes are:

1. to prepare for a port to Unix by better isolating the platform-specific code and removing some needlessly Windows-specific code and symbol names
2. to separate more Common Graphics code into optional modules so that standalone Common Graphics apps can be smaller
3. to separate cleanly the IDE from Common Graphics so that Common Graphics can be used in a development Lisp that does not contain the IDE.

Achieving these purposes has required an unusually large number of backwardly incompatible changes. On the other hand, the large majority of these changes involve relatively obscure functionality (to which less attention is usually paid), and most other incompatibilities consist of symbol-name or package changes that should be straightforward to address in application code. Nevertheless, we hope that the volume of release notes is not overwhelming.

The IDE has been cleanly separated from Common Graphics for this release. As part of that, some symbols that were exported from the `cg` package are now exported from the new `ide` package instead. Also, the `cg:*system*` object has been divided into `cg:*system*` and `ide:*ide-system*`, and there are separate configuration objects for CG and the IDE, returned by the expressions `(cg:configuration cg:*system*)` and `(cg:configuration ide:*ide-system*)`. Most of the configuration properties have moved to the IDE configuration object, and so programmatic calls to return or set these properties need to be changed to retrieve the configuration object from `ide:*ide-system*` rather than from `cg:*system*`. For example, code that uses a form such as:

```
(setf (scroll-while-tracing (configuration *system*)) t)
```

must change it to:

```
(setf (scroll-while-tracing (configuration *ide-system*)) t)
```

While there are more IDE than CG configuration options, the CG options are the ones that were kept backwardly compatible because they are the ones that may be used in a standalone application, and therefore are the ones that are more likely to appear in existing user code. IDE options are typically changed interactively only.

The following lists identify the symbols that have been moved from the `cg` package to the `ide` package. The first list is configuration options, with some special notes. The second list is other symbols. Finally, there is a list of new symbols in the new `ide` package.

List1: configuration options previously in `cg` but now in `ide` package

open-files-in-gnu-emacs

allow-component-overlap	ask-before-auto-saving	backtrace-print-circle
backtrace-print-length	backtrace-print-level	cg-tree-font
class-graph-font	class-graph-initial-depth	close-inactive-listeners
code-file-pretty-print-columns	comment-indent	context-sensitive-default-path
debug-font	debug-history-limit	display-console
display-extended-toolbar	display-form-grid	display-status-bar
display-toolbars	editor-font	
editor-mode	find-again-after-replace	grid-x-spacing
grid-y-spacing	incremental-search	initial-package
initial-search-directories	inspector-name-font	inspector-style

inspector-value-font	lisp-message-print-level	lisp-message-print-length
maximize-ide-background-window	maximum-symbol-completion-choices	min-pixels-between-widgets
new-project-show-editor	new-project-show-form	new-project-show-project-manager
open-project-action	pretty-printer	printer-font
profiler-included-node-types	query-exit	recent
recent-limit	save-options-on-exit	save-options-to-user-specific-file
scroll-while-tracing	show-dialog-on-compiler-warnings	snap-to-components
snap-to-grid	standard-toolbar-icons	start-in-allegro-directory
symbol-completion-searches-all-packages	use-ide-background-window	use-private-html-browser
warn-on-no-action-taken	window-configurations	

Notes

- **cg:prompt** has been removed.
- **ide-prompt** replaces it.

Below are the other symbols that had been exported from the `cg` package and that are now exported from the `ide` package instead. Most pertain to projects or form windows, which exist only in the IDE.

In general, application code would not have called these functions since they were already unavailable in a standalone app. An exception to this rule may be **finder-function** and **maker-function**, where the documentation implies their use by applications. Since the project from which a standalone app is generated no longer exists in the standalone app (for size and general design reasons), these functions may no longer be called in the application code to find the function to call to make a module's window. Instead, the application code should directly call the finder function or maker function. (The new function **main-window-maker** may be called in a standalone app though.)

List 2: other symbols previously in `cg` but now in `ide` package

add-to-component-toolbar	after-functions	all-files
all-projects	autoload-warning	before-functions
build-flags	<code>build-module</code>	build-project
close-current-project	compilation-unit	compile-project
comtab-report	current-project	default-command-line-arguments
defdefiner	define-project	distributed-files
eval-in-listener-thread	exit-tests	fanatical-followers
find-module	find-project	find-window-configuration
finder-function	followers	form-file
<code>form-module</code>	form-package-name	<code>form-pane</code>
full-compile-project	full-recompile-for-runtime-conditionalizations	graph-child-windows
icon-file	ide-evaluator-listener	<code>*ide-startup-hook*</code>
ignore-package-name-case	in-break	include-flags
keep-configuration	libraries	<code>library-module</code>
lisp-message	<code>*lisp-message-window*</code>	lisp-warning
load-project	loaded	main-form
main-module	maker-function	<code>module</code>
module-p	modules	new-space-size
old-space-size	on-initialization	on-restart
open-project	partner-types	partners
process-trace-color	<code>project (gf) project (class)</code>	project-file
<code>project-module</code>	project-package	project-package-name
projects	property-starter-code	remove-from-component-toolbar
restore-window-configuration	running-form	running-window

runtime-build-option	runtime-modules	splash-file
start-ide	*starting-ide*	subobjects-with-partners
trace-format	verbose	with-trace-color

List 3: new symbols in the ide package

additional-build-lisp-image-arguments	ask-for-action-at-ide-startup	close-project-close-editor-buffers
display-help	distribution-directories	file-dialog-source-types
ide-configuration	ide-implementation-version	*ide-is-running*
ide-prompt	*ide-system*	new-project-create-form
open-project-show-files-in-editor	open-project-show-project-manager	project-parent-directory
query-os-exit		

Appendix C: The reorganization of the CG class structure

Some classes have been removed, some renamed, and some new ones have been provided. Applications are unlikely to have used the renamed or removed classes as they were all non-instantiable. The details of the changes are as follows:

- **widget-window renamed os-widget-window, new class named widget-window created.** The exported (in 6.2) class `widget-window` has been renamed to `os-widget-window`. And a new class has been named `widget-window`. So the `widget-window` class is now the class of windows for all widgets, and its subclasses are `os-widget-window` and `lisp-widget-window`. The formerly unexported symbol `windows-widget` has been renamed to `os-widget` and is exported.

To repeat something said in [Section 8.1 Non-backward-compatible changes in Common Graphics](#), when defining a custom `lisp-widget`, you create a window class for the window holding the widget. In earlier releases, this window class should be a subclass of both `lisp-widget-window` and `lisp-widget-top-window`. In 7.0, the only superclass should be `lisp-widget-top-window` (which is now a subclass of `lisp-widget-window`).

- **New class `os-widget`.** `os-widget` is the non-instantiable superclass of dialog items provided by the underlying window system or operating system.
- **Window class removed.** The class named `window` has been removed: it was just a mixin to group `basic-pane` and `menu` together, though those classes are not significantly related.
- **Windows-screen-device has been renamed `screen`.** `windows-screen-device` no longer names a class. The new name is `screen`.
- **New class `screen-stream`.** The new `screen-stream` class groups together all streams that display on the screen, which includes all windows (see `basic-pane`) and the `screen` itself.
- **New class `lisp-edit-window`.** The new `lisp-edit-window` class is a `frame-with-single-child` class similar to the existing `text-edit-window` class, except that the **frame-child** window will be a `lisp-edit-pane` rather than a `text-edit-pane`.
- **Various other non-instantiable classes are now named with exported symbols.** Several other non-instantiable classes are newly exported simply because they are superclasses of instantiable `dialog-item` and `widget-window` classes, and it may be helpful to document the meaning of each class. These include `scroll-bar-mixin`, `scroll-bar-pane-mixin`, `scroll-bar`, `scroll-bar-pane`, `text-widget`, `text-widget-pane`, `item-list`, `item-list-pane`, `toggling-widget`, `toggling-widget-pane`, `picture-widget`, and `picture-widget-pane`.

Appendix D: Details of separating DDE code from Common Graphics

One of the significant changes in Common Graphics from 6.2 to 7.0 is the separation of DDE code from the rest of Common Graphics. There was no logical connection between DDE and Common Graphics, other than both are available on Windows only. Common Graphics thus served as the module for most Windows-specific functionality. With the general reorganization of Common Graphics in release 7.0, DDE is now separate.

An application may now use the new "dde" module without loading any of Common Graphics. This means that the exported symbols are no longer exported from the `cg` package, and are now exported only from the `dde` package.

If you have code which uses DDE, make sure of the following:

- If any DDE symbols (listed below) are package-qualified `cg:`, you must change the qualification to `dde:`.
- If you have a package that uses the `cg` package, and you refer to DDE symbols while it is the current package, you must either have your package also use the `dde` package or you must package qualify the symbols with `dde:` (using the `dde` package is the expected thing to do).
- If you use DDE code, be sure to require the `dde` module. In your code, you can add `(require :dde)`. If you build images with **build-lisp-image** or **generate-application**, be sure `:dde` is included in the list of files to include in the application.

The name of the document that describes the DDE facility has also been changed. It was `cg-dde.htm` and was in the `doc/cg/` directory. It is now **dde.htm** and is in the `doc/` directory.

Several symbols have been added, naming new functionality, as described below. The symbol `*show-dde-warnings*` has been removed. It has been replaced by `*generate-dde-messages*`.

The symbols moved from the `cg` to the `dde` package are:

Symbols naming operators

active-client-ports	active-server-ports	answer-request
case-sensitive-dde	close-dde	close-port
close-server	convert-returned-dde-buffer	convert-returned-dde-string
dde-info	dde-message	execute-command
free-item	open-port	open-server
port-application	port-name	port-open-p
port-topic	post-advice	receive-advice
send-command	send-request	send-value
server-active-p	service-name	service-topics
systems		

Symbols naming variables

<code>*case-sensitive-dde*</code>	<code>*service-name*</code>	<code>*service-topics*</code>
<code>*systems*</code>		

Symbols naming classes

<code>client-port</code>	<code>dde-info</code>
--------------------------	-----------------------

New symbols in the dde package

These symbols have been added in release 7.0:

- **dde-message**
 - **dword-list-from-dde-buffer**
 - ***generate-dde-messages***
 - **string-from-dde-buffer**
-

Appendix E: Functionality/symbols removed or renamed in Common Graphics

The following Common Graphics functionality is no longer supported. All items are operators unless otherwise noted. We do not list symbols moved from one package to another (from `cg` to `dde` or from `cg` to `ide`). There are individual release notes (in [Section 8.1 Non-backward-compatible changes in Common Graphics](#)) for most of the items listed here.

1. **cg:application-type**
2. **cg:brief-comtab**
3. **cg:button-p** (use **typep** instead)
4. **cg:check-fixnums**
5. **cg:clipboard**
6. **cg:compile-unsaved-form**
7. **cg:device-bitmap**
8. **cg:emacs-comtab**
9. **cg:eof-p**
10. **cg:erase** (variable)
11. **cg:files-to-list-box**
12. **cg:find-widget** (use **find-component** instead)
13. **cg:host-comtab**
14. **ide:ignore-package-name-case** (no-op function still exists)
15. **cg:invert** (variable)
16. **cg:list-to-tabbed-string**
17. **cg:outline-item** (use **find-outline-item**)
18. **cg:paint** (variable)
19. **cg:pathname-string-from-directory-list-box**
20. **cg:pending-message**
21. **cg:port-name** (renamed **printer-port-name**)
22. **cg:pretty-printed-object**
23. **cg:query-end-windows-session** (replaced by **os-exit-request**)
24. **cg:raw-text-edit-comtab**
25. **cg:reflect-texture-in-x** (use **reflect-pixmap-in-x** instead)
26. **cg:reflect-texture-in-y** (use **reflect-pixmap-in-y** instead)
27. **cg:return-value-to-windows**
28. **cg:set-dialog-item-title** (use **(setf title)** instead)
29. **cg:set-dialog-item-value** (use **(setf value)** instead)
30. **cg:size**
31. **cg:stream-device**

32. **cg:stream-location**
 33. **cg:subdirectories-to-list-box**
 34. **cg:system-color**
 35. **cg:tabbed-string-to-list** (use **delimited-string-to-list** instead)
 36. **cg:text-edit-comtab**
 37. **cg:update-text-widget**
 38. **cg:web-page-contents**
 39. **cg:with-room**
 40. **cg:*after-session-init-functions-hook*** (variable)
 41. **cg:*check-cg-args*** (variable)
 42. **cg:*edit-allowed-types*** (variable)
 43. **cg:*show-dde-warnings*** (variable, use `dde: *generate-dde-messages*`)
-

Appendix F: Common Graphics compatibility with `aclpc 3.0.2` has been removed

There is no longer a compatibility module for the old Common Graphics function names from the 3.0.2 release of Allegro CL on Windows. This module had supplied stub functions that automatically translated calls to the old functions into calls to the current functions, which mostly were given shorter names in the 5.0 release. If your application still calls any of the 3.0.2 functions, the source code must now be converted to use the current function names.

The file `cg-obsolete.cl` (in the main Allegro directory) contains an association list with several hundred entries that map from the old names to the new; you could either search this file textually for Common Graphics function names from your code that are no longer found, or perhaps do a lisp read on the file in order to search the list programmatically. Each entry in the alist is a list of an obsolete symbol followed by the corresponding current symbol or form; in most cases the new symbol or form will substitute in directly, though in some cases the new symbol must be used somewhat differently, as described in the current documentation for that symbol. A few of the entries have a third member, in which case the second member corresponds directly to the old symbol, and the third member is something that is preferable to that.

Also, the backward compatibility that allowed creating an IDE project from interactively-built dialogs that were last saved in `aclpc` version 3.0.2 or before has been removed.

Appendix F.1 The `aclwin` and `aclffi` compatibility modules are no longer loaded by Common Graphics

Common Graphics no longer uses the `aclwin` and `aclffi` modules, which exist for backward compatibility with the 3.0.2 base lisp on Windows. A Common Graphics application that calls functions in those deprecated modules directly would need to either load the module(s) explicitly or (preferably) convert the calls to current equivalents. In particular, calls to the `aclffi` functions `cref`, `cset`, and `csets` should be converted to `fslot-value` and `(setf fslot-value)`, and `allocate` and `callocate` calls should be converted to `allocate-fobject`. Symbols from the `aclwin` module that an application may have used include `win32p` (no longer needed since Allegro has not run on 16-bit Windows for some time) and the `for` macro (use the `loop` macro instead, which is similar but not identical).

Appendix G: The effects of the new, longer array implementation on `def-foreign-call` and `def-foreign-type`

The new array implementation is discussed in the **Arrays and short arrays** section in **implementation.htm**. In brief, standard Common Lisp arrays now can be significantly larger than in earlier releases, while the new short arrays implement the old arrays (the same size limitations but also the same type codes and structure).

In this discussion, ``array'` refers to the newly-implemented arrays, while ``short-array'` refers to the old implementation, preserved as short arrays in 7.0.

`short-array` is not a subtype of `array`. Therefore, it is possible there may be problems when the system expects an array and receives a short-array. In this appendix, we discuss these problems as they affect **`def-foreign-call`** and **`def-foreign-type`**.

Affect on `def-foreign-call`

Foreign calls are made with arrays as arguments by passing the address of the first value. In the new implementation, simple-arrays always have exactly the same first element offset (but some short-arrays are aligned to the next higher word boundary so that the elements within are naturally aligned). This sometimes-difference between arrays and short-arrays poses an extra burden on the foreign function interface, in that the arrays must be distinguished between themselves at runtime.

It is now possible to declare an argument to be passed to a foreign function (in **`def-foreign-call`**) to be a `(short-simple-array ... (*))` and the interface will generate code as it did before for normal arrays, passing the address of the first argument.

For 7.0, a declaration of `(simple-array ... (*))` actually generates code that tests at runtime whether the argument is a short-array or a normal array. So in effect, a short-simple-array passed in as if it were a normal simple-array will be properly passed.

Note that with this setup, if argument checking is specified and a short-array is passed in, the check will fail, because a short-array is not a subtype of simple-array. But, if you suppress this argument checking, the interface will pass either array correctly.

However, programmers are urged to provide correct declarations and pass the correct type of array even though 7.0 allows sloppiness. (In a future release, we anticipate adding a `(dual-simple-array ... (*))` declaration to the direct-call foreign interface, telling the system that either a short-array or a regular array will be passed in.)

Affect on foreign types

With the advent of the long array implementation, the foreign types system makes a distinction between regular vectors and short-vectors. It must do so, because the layout of each is different.

All foreign-type interface functions which accept the `:lisp` allocation type (such as **`allocate-fobject`**, **`fslot-value-typed`** and its inverse, **`fslot-address-typed`**, **`describe-fobject`**, and **`def-foreign-type`**) have now been enhanced to accept a `:lisp-short` value for the allocation type. Since the basic simple-array type has changed its implementation, the `:lisp` allocation type now describes the regular arrays, and the new `:lisp-short` allocation type describes short-arrays. If the allocation is not given, and can be figured out (e.g. a null allocation argument to **`fslot-value-typed`**, or a call to **`fslot-value`**) then the object is tested to see if it is a long or a short vector, and the allocation type is set based on the test.

Because of the change in layout between the older short-vector types and the new (long) vector types, the foreign type allocation must match the object. This is not likely to be a problem, if nothing special is done to make short-arrays.