Release Notes and Installation Guide

Version 8.0

Copyright and Trademarks

Release Notes and Installation Guide

Version 8.0

December 2021

Address	Telephone	Fax
LispWorks Ltd St. John's Innovation Centre Cowley Road Cambridge CB4 0WS England	From North America: 877 759 8839 (toll-free) From elsewhere: +44 1223 421860	From North America: 617 812 8283 From elsewhere: +44 870 2206189

www.lispworks.com

Contents

1 Introduction 8

1.1 LispWorks Editions 8
1.2 LispWorks for Mobile Runtime 9
1.3 Evaluation quick guide 9
1.4 Further details 10
1.5 About this Guide 10

2 Installation on macOS 12

2.1 Choosing the Graphical User Interface 12
2.2 Documentation 12
2.3 Software and hardware requirements 12
2.4 Installing LispWorks for Macintosh 13
2.5 Starting LispWorks for Macintosh 16
2.6 Uninstalling LispWorks for Macintosh 17
2.7 Upgrading the LispWorks Edition 17

3 Installation on Windows 18

3.1 Documentation 18
3.2 Installing LispWorks for Windows 18
3.3 Uninstalling LispWorks for Windows 20
3.4 Upgrading the LispWorks Edition 20
3.5 Upgrading to 64-bit LispWorks 20

4 Installation on Linux 21

4.1 Software and hardware requirements 21
4.2 License agreement 22
4.3 Software delivery and installer formats 22
4.4 Installing LispWorks for Linux 23
4.5 LispWorks looks for a license key 26
4.6 Running LispWorks 27
4.7 Configuring the image 28
4.8 Printable LispWorks documentation 28
4.9 Uninstalling LispWorks for Linux 28
4.10 Upgrading the LispWorks Edition 28
4.11 Upgrading to 64-bit LispWorks 28

5 Installation on x86/x64 Solaris 29

5.1 Software and hardware requirements 29
5.2 Software delivery and installer format 30
5.3 Installing LispWorks for x86/x64 Solaris 30
5.4 LispWorks looks for a license key 32
5.5 Running LispWorks 32
5.6 Configuring the image 32
5.7 Printable LispWorks documentation 33
5.8 Uninstalling LispWorks for x86/x64 Solaris 33
5.9 Upgrading the LispWorks Edition 33
5.10 Upgrading to 64-bit LispWorks 33

6 Installation on FreeBSD 34

6.1 Software and hardware requirements 34
6.2 License agreement 35
6.3 Software delivery and installer format 35
6.4 Installing LispWorks for FreeBSD 35
6.5 LispWorks looks for a license key 37
6.6 Running LispWorks 37
6.7 Configuring the image 38
6.8 Printable LispWorks documentation 38
6.9 Uninstalling LispWorks for FreeBSD 38
6.10 Upgrading the LispWorks Edition 38
6.11 Upgrading to 64-bit LispWorks 38

7 Installation of LispWorks for Mobile Runtime 39

7.1 Installing LispWorks for Android Runtime 39
7.2 Installing LispWorks for iOS Runtime 39

8 Configuration on macOS 40

8.1 Introduction 40
8.2 License keys 40
8.3 Configuring your LispWorks installation 40
8.4 Saving and testing the configured image 42
8.5 Initializing LispWorks 44
8.6 Loading CLIM 2.0 44
8.7 The Common SQL interface 45
8.8 Common Prolog and KnowledgeWorks 46

9 Configuration on Windows 47

9.1 Introduction 47
9.2 License keys 47
9.3 Configuring your LispWorks installation 47

9.4 Saving and testing the configured image 48
9.5 Initializing LispWorks 50
9.6 Loading CLIM 2.0 50
9.7 The Common SQL interface 51
9.8 Common Prolog and KnowledgeWorks 51
9.9 Runtime library requirement on Windows 52

10 Configuration on Linux, x86/x64 Solaris & FreeBSD 53

10.1 Introduction 53
10.2 License keys 53
10.3 Configuring your LispWorks installation 54
10.4 Saving and testing the configured image 55
10.5 Initializing LispWorks 56
10.6 Loading CLIM 2.0 56
10.7 The Common SQL interface 57
10.8 Common Prolog and KnowledgeWorks 58
10.9 Documentation on x86/x64 Solaris and FreeBSD 58

11 Troubleshooting, Patches and Reporting Bugs 59

11.1 Troubleshooting 59
11.2 Troubleshooting on Windows 61
11.3 Troubleshooting on macOS 61
11.4 Troubleshooting on Linux 61
11.5 Troubleshooting on x86/x64 Solaris 62
11.6 Troubleshooting on FreeBSD 63
11.7 Troubleshooting on X11/Motif 63
11.8 Updating with patches 64
11.9 Reporting bugs 66
11.10 Transferring LispWorks to a different machine 69

12 Release Notes 71

12.1 Keeping your old LispWorks installation 71
12.2 Updating your code for LispWorks 8.0 71
12.3 Platform support 71
12.4 GTK+ window system 73
12.5 New CAPI features 74
12.6 Other CAPI and Graphics Ports changes 76
12.7 More new features 76
12.8 IDE changes 83
12.9 Editor changes 85
12.10 Foreign Language interface changes 86
12.11 Objective-C changes 87
12.12 Common SQL changes 87
12.13 KnowledgeWorks changes 88

12.14 Application delivery changes 88
12.15 Other changes 89
12.16 Documentation changes 90
12.17 Known Problems 91
12.18 Binary Incompatibility 92

Index

1 Introduction

1.1 LispWorks Editions

LispWorks is available in several product editions on desktop platforms.

The main differences between the editions are outlined below. Further information can be found at:

www.lispworks.com/products

1.1.1 Personal Edition

LispWorks Personal Edition allows you to explore a fully-enabled Common Lisp programming environment and to develop small- to medium-scale programs for personal and academic use. It includes:

- Native graphical IDE.
- Full Common Lisp compiler.
- COM/Automation API on Microsoft Windows.

LispWorks Personal Edition has several limitations. These are:

- A heap size limit
- A time limit of 5 hours for each session.
- The functions **`save-image`**, **`deliver`**, and **`load-all-patches`** are not available.
- Initialization files are not available.
- HobbyistDV, Professional and Enterprise Edition module loading is not included.

LispWorks Personal Edition has no license fee. Download it from:

www.lispworks.com/downloads

1.1.2 Hobbyist Edition

LispWorks 8.0 Hobbyist Edition is available to individual licensees for non-commercial and non-academic use. It is a fully-functional Common Lisp IDE without most of the limitations of the Personal Edition:

- No heap size limit.
- No session time limit.
- The functions **`save-image`** and **`load-all-patches`** are available.
- Initialization files are available.

HobbyistDV, Professional and Enterprise Edition module loading is not included. In particular, the function **`deliver`** is omitted so runtimes cannot be generated.

1.1.3 HobbyistDV Edition

LispWorks 8.0 HobbyistDV Edition is available to individual licensees for non-commercial and non-academic use. It has all the features of the Hobbyist Edition plus:

- The function **`deliver`** allowing generation of non-commercial end-user applications and libraries.

1.1.4 Professional Edition

LispWorks 8.0 Professional Edition includes all the features of the HobbyistDV Edition plus:

- Fully supported commercial product.
- Delivery of commercial end-user applications and libraries.
- CLIM 2.0 on X11/Motif and Windows.
- 30-day free "Getting Started" technical support.

1.1.5 Enterprise Edition

LispWorks 8.0 Enterprise Edition provides further support for the software needs of the modern enterprise. It has all the features of the Professional Edition plus:

- Database access through the Common SQL interface.
- Portable distributed computing through CORBA.
- Expert systems programming through KnowledgeWorks and embedded Prolog compiler.

On most platforms you can choose either the 32-bit or 64-bit implementation of LispWorks. These implementations are licensed separately.

1.2 LispWorks for Mobile Runtime

LispWorks for Android Runtime and LispWorks for iOS Runtime are new products which you can use to build LispWorks runtimes for inclusion in mobile apps.

1.3 Evaluation quick guide

If you are evaluating LispWorks, then the following notes might prove to be useful.

- LispWorks support (**`lisp-support@lispworks.com`**) will be happy to answer any issues you have.
- The LispWorks distribution contains various examples demonstrating various features of LispWorks. All the examples are in the directory "examples" inside the LispWorks installation.

  You can find this directory by evaluating the following in a LispWorks Listener:

  ```
  (example-file "")
  ```

  Each example contains comments that explain what it demonstrates.

  In many cases it is convenient to copy the example and modify it to do what you want, rather than writing your own code from scratch.

- If you encounter an error that is not obviously a bug in your code, it is always best to produce a full bug report as described in **11.9.3 Generate a bug report template**. This will speed up the resolution of the issue.
- If you have performance issues, you should use **`room`**, **`extended-time`** and **`profile`** to narrow the problem. See the *LispWorks® User Guide and Reference Manual* for details of these diagnostic functions and macros. You should also report it to LispWorks support, as LispWorks is efficient in general and we do not expect performance problems.

1.4 Further details

For further information about LispWorks products visit:

www.lispworks.com

To purchase LispWorks please follow the instructions at:

www.lispworks.com/buy

1.5 About this Guide

This document is an installation guide and release notes for LispWorks 8.0 on macOS, Windows, Linux, x86/x64 Solaris, FreeBSD platforms and LispWorks for Mobile Runtime. It also explains how to configure LispWorks to best suit your local conditions and needs.

This guide provides instructions for installing and loading the modules included with each Edition or add-on product.

Unless explicitly mentioned, instructions in this manual refer to the Hobbyist, HobbyistDV, Professional and Enterprise Editions, rather than the Personal Edition or LispWorks for Mobile Runtime which are distributed separately.

1.5.1 Installation and Configuration

Chapters **2 Installation on macOS** -**6 Installation on FreeBSD** explain in brief and sufficient terms how to complete a LispWorks installation on macOS, Windows, Linux, x86/x64 Solaris or FreeBSD. Choose the chapter for your platform: **2 Installation on macOS**, **3 Installation on Windows**, **4 Installation on Linux**, **5 Installation on x86/x64 Solaris**, or **6 Installation on FreeBSD**.

Chapter **7 Installation of LispWorks for Mobile Runtime** briefly mentions installation of LispWorks for Mobile Runtime.

Chapters **8 Configuration on macOS**-**10 Configuration on Linux, x86/x64 Solaris & FreeBSD** explain in detail everything necessary to configure, run, and test LispWorks 8.0. Choose the chapter for your platform: **8 Configuration on macOS**. **9 Configuration on Windows**, or **10 Configuration on Linux, x86/x64 Solaris & FreeBSD**. This also includes sections on initializing LispWorks and loading some of the modules. You should have no difficulty configuring, running, and testing LispWorks using these instructions if you have a basic familiarity with your operating system and Common Lisp.

1.5.2 Troubleshooting

Chapter **11 Troubleshooting, Patches and Reporting Bugs** discusses other issues that may arise when installing and configuring LispWorks. It includes a section that provides answers to problems you may have encountered, sections on the LispWorks patching system (used to allow bug fixes and private patch changes between releases of LispWorks), and details of how to report any bugs you encounter.

1.5.3 Release Notes

Chapter **12 Release Notes** highlights what is new in this release and special issues for your consideration.

2 Installation on macOS

This chapter is an installation guide for LispWorks 8.0 (64-bit) for Macintosh. **8 Configuration on macOS** discusses post-installation and configuration in detail, but this chapter presents the instructions necessary to get LispWorks up and running on your system.

2.1 Choosing the Graphical User Interface

LispWorks for Macintosh supports three different graphical interfaces. Most users choose the native macOS GUI, but you can use the X11 GUI option instead, which supports both GTK+ and Motif. (Motif is deprecated, though.)

Different executables and supporting files are supplied for the two GUI options. You need to decide at installation time which of these you will use, or you can install support for both. If you install just one GUI option and later decide to install the other, you can simply run the installer again.

LispWorks for Macintosh Personal Edition supports only the native macOS GUI.

2.2 Documentation

The LispWorks documentation set is included in two electronic formats: HTML and PDF. You can chose whether to install it as described in **2.4 Installing LispWorks for Macintosh**.

The HTML format can be used from within the LispWorks IDE via the **Help** menu. You will need to have a suitable web browser installed. You can also reach the HTML documentation via the alias
`LispWorks 8.0/HTML Documentation.htm`. If you choose not to install the documentation, you will not be able to access the HTML Documentation from the LispWorks **Help** menu.

The PDF format is suitable for printing. Each manual in the documentation set is presented in a separate PDF file in the LispWorks library under `manual/offline/pdf`. The simplest way to locate these PDF files is the alias
`LispWorks 8.0/PDF Documentation`. To view and print these files, you will need a PDF viewer such as Preview (standard on macOS) or Adobe® Reader® (which can be downloaded from the Adobe website at **www.adobe.com**).

2.3 Software and hardware requirements

LispWorks 8.0 supports Macintosh computers containing Intel CPUs.

An overview of system requirements is provided in the table **System requirements on macOS**. The sections that follow discuss any relevant details.

System requirements on macOS

Product	Hardware Requirements	Software Requirements
LispWorks (64-bit) for Macintosh	Intel or Apple silicon processor. 338MB of disk space including documentation	macOS version 10.6.x or higher for Intel and 11.5.x or higher for Apple silicon. GTK+ 2 (version 2.4 or higher) if you want to run the GTK+ GUI. Open Motif 2.3 and Imlib2 1.4.9 if you want to run the deprecated Motif GUI.

2.4 Installing LispWorks for Macintosh

2.4.1 Main installation and patches

The LispWorks 8.0 installer contains each of the Editions. Additionally, there may be a patch installer which upgrades LispWorks to version 8.0.x. You need to complete the main installation before adding patches.

2.4.2 Information for Beta testers

Users of LispWorks 8.0 Beta should completely uninstall it (including any patches added to the beta installation) before installing LispWorks 8.0.

See **2.6 Uninstalling LispWorks for Macintosh** for instructions.

2.4.3 Information for users of previous versions

You can install LispWorks 8.0 in the same location as LispWorks 7.1 or previous versions. If you always choose the default install location, a new folder named **`LispWorks 8.0 (64-bit)`** will be created alongside the other versions.

2.4.4 Launch the LispWorks installer

The LispWorks installer is a **`pkg`** file, with the following name:

`LispWorks80-64bit_Installer.pkg` (64-bit Lispworks)

`LispWorksPersonal80_Installer.pkg` (LispWorks Personal Edition)

To install LispWorks, launch this file, which should run the macOS Installer application. If this does not happen, right-click on th file and choose **`Open With > Installer`**.

The Introduction page should be displayed. Click **Continue** to go to the next step.

2.4.5 The Read Me

The Read Me presented next by the installer is a plain text version of this *Release Notes and Installation Guide*.

2.4.6 The License Agreement

Check the license agreement, then click **Continue**. You will be asked if you agree to the license terms. Click the **Agree** button only if you accept the terms of the license. If you click **Disagree**, then the installer will not proceed.

2.4.7 Install Location

All the files installed with LispWorks are placed in the LispWorks folder, which is named **`LispWorks 8.0 (64-bit)`**, or **`LispWorks Personal 8.0`** depending on which edition you are installing. The LispWorks folder is placed in the main **`Applications`** folder for use by all users.

Note: The **`Applications`** folder may display in the Finder with a name localized for your language version of macOS.

2.4.8 Choose your installation type

The default Standard Install includes the native macOS GUI and the documentation, but you can also customize the install, for examle to select the X11 GUI option.

Different executables and supporting files are supplied for the two GUI options. If you install just one of these and later decide to install the other, you can simply run the installer again.

2.4.8.1 The native macOS GUI

If you simply want to install LispWorks for the native macOS GUI, and the documentation, click **Install**.

2.4.8.2 The X11 GTK+ and Motif GUIs

If you want to use LispWorks with either of the alternative X11 GUIs, click **Customize** and select the option **LispWorks with X11 IDE** under **Extra items**.

The default X11 GUI is GTK+. Motif is also available, but is deprecated. You can select Motif at run time.

Note: to run LispWorks with an X11 GUI, you will need both of these installed:

- An X server such as Apple's X11.app, available at **www.apple.com**.
- One of GTK+ 2 (version 2.4 or higher) or Open Motif 2.3.

If you use Open Motif, you will also need Imlib2 version 1.4.9 or later.

None of these are required at the time you install LispWorks, however.

The X11 GUIs are not available for the Personal Edition.

2.4.8.3 The Documentation

If you use the Standard Install the documentation will be installed.

If you do not wish to install the documentation, click **Customize** and uncheck the **LispWorks documentation** option under **Standard items**.

2.4.9 Installing and entering license data

Now click **Install**.

You will be prompted for an administrator's name and password.

If you are not installing the LispWorks Personal Edition, then enter your serial number and license key when the installer asks for these details.

Your license key will be supplied to you in email from Lisp Support or Lisp Sales.

If you have problems with your LispWorks license key, send it to **`lisp-keys@lispworks.com`**, showing the complete output after you enter it, preferably with a screenshot.

2.4.10 LispWorks is added to the Dock

The installer adds LispWorks to the Dock.

2.4.11 Finishing up

You should now see a message confirming that installation of LispWorks was successful. Click the **Close** button.

Note: LispWorks needs to be able find its library at run time and therefore the LispWorks installation should not be moved around piecemeal. If you must move it, move the entire LispWorks installation folder. If you simply want to run LispWorks from somewhere more convenient, then consider adding an alias.

2.4.12 Installing Patches

After completing the main installation of LispWorks, ensure you install the latest patches which are available for download at **www.lispworks.com/downloads/patch-selection.html**. Patch installation instructions are in the README file accompanying the patch download.

2.4.13 Obtaining X11 GTK+

LispWorks does not provide GTK+ libraries, so you need to install third-party libraries, such as:

- the gtk+2 package from the Fink Project at **www.finkproject.org**, or:
- the gtk2 package from MacPorts at **www.macports.org**.

Note: you need the x11 gtk2 libraries, not GTK-OSX (Quartz).

2.4.14 Obtaining Open Motif and Imlib2

LispWorks 8.0 for Macintosh on X11/Motif requires Open Motif 2.3 and Imlib2 1.4.9.

The Open Motif library for LispWorks is **`/usr/local/lib/libXm.4.dylib`**.

Lisp Support can supply suitable Motif and Imlib2 libraries if you need them.

Note: The Motif GUI is deprecated. A GTK+ GUI is available.

2.5 Starting LispWorks for Macintosh

2.5.1 Start the native macOS LispWorks GUI

Assuming you have installed this option, you can now start LispWorks with the native macOS GUI by double-clicking on the LispWorks icon in the LispWorks folder.

Note: The LispWorks folder is described in **2.4.7 Install Location**.

If you added LispWorks to the Dock during installation, you can also start LispWorks from the Dock. If you did not add LispWorks to the Dock during installation, you can add it simply by dragging the LispWorks icon from the Finder to the Dock.

If you want to create a LispWorks image that does not start the GUI automatically, then see **8.4.5 Saving a non-windowing image** (this option is not available in the Personal Edition).

See **8.3 Configuring your LispWorks installation** for more information about configuring your LispWorks image for your own needs.

Note: for the Personal Edition, the folder name and icon name are LispWorks Personal.

2.5.2 Start the GTK+ LispWorks GUI

Assuming you have installed the "LispWorks with X11 IDE" option, and that you have X11 running and GTK+ installed, you can now start LispWorks with the GTK+ GUI.

Follow this session in the X11 terminal for 64-bit LispWorks (the filenames will be slightly different for 64-bit LispWorks):

```
bash-3.2$ cd "/Applications/LispWorks 8.0 (64-bit)"
bash-3.2$ ./lispworks-8-0-0-macos64-universal-gtk
; Loading text file /Applications/LispWorks 7.1 (64-bit)/Library/lib/8-0-0-0/private-patches/load.l
isp
LispWorks(R): The Common Lisp Programming Environment
Copyright (C) 1987-2021 LispWorks Ltd.  All rights reserved.
Version 8.0.0
Saved by LispWorks as lispworks-8-0-0-amd64-darwin-gtk, at 02 Aug 2021 15:21
User lw on machine.lispworks.com
; Loading text file /Applications/LispWorks 8.0 (64-bit)/Library/lib/8-0-0-0/config/siteinit.lisp
;  Loading text file /Applications/LispWorks 8.0 (64-bit)/Library/lib/8-0-0-0/private-patches/load.
lisp
; Loading text file /Users/lw/.lispworks
```

The LispWorks GTK+ IDE should appear.

See **8.3 Configuring your LispWorks installation** for more information about configuring your LispWorks image for your own needs.

2.5.3 Start the Motif LispWorks GUI

Assuming you have installed the "LispWorks with X11 IDE" option, and that you have X11 running and Motif and Imlib2 installed, you can use LispWorks with the Motif GUI.

You first must load the Motif GUI into the supplied **`lispworks-8-0-0-macos64-universal-gtk`** image, by:

```
(require "capi-motif")
```

This loads the necessary module and makes Motif the default library for CAPI.

Then you can start the LispWorks IDE by calling the function **`env:start-environment`**. You might want to save an image with the **`"capi-motif"`** module pre-loaded: do this with a **`save-image`** script containing:

```
(require "capi-motif")
```

2.6 Uninstalling LispWorks for Macintosh

To uninstall LispWorks you should run the file **`uninstall.command`** in the LispWorks folder. This must be run as an administrator user.

2.7 Upgrading the LispWorks Edition

Some LispWorks features such as Delivery, Common SQL and KnowledgeWorks are not available in all Editions. You can add these features by upgrading.

After purchasing your upgrade from **lisp-sales@lispworks.com**, select **Help > Register...** and enter your new license key.

3 Installation on Windows

This chapter is an installation guide for LispWorks 8.0 (32-bit) for Windows and LispWorks 8.0 (64-bit) for Windows. **9 Configuration on Windows** discusses post-installation and configuration in detail, but this chapter presents the instructions necessary to get LispWorks up and running on your system.

3.1 Documentation

The LispWorks documentation set is available in two electronic forms: HTML and PDF. You can choose whether to install either of these.

If you install the HTML documentation, then it can be used from within the the LispWorks IDE via the **Help** menu. It is also available from the Windows 7 **Start** menu under **Start > All Programs > LispWorks 8.0 > HTML Documentation** or on the Windows 8 start screen.

The PDF format is suitable for printing. Each manual in the documentation set is presented in a separate PDF file, available from the **Start** menu under **Start > All Programs > LispWorks 8.0 > PDF Documentation**. To view and print these files, you will need a PDF viewer such as Adobe® Reader®. If you do not already have this, it can be downloaded from the Adobe website.

3.2 Installing LispWorks for Windows

3.2.1 Main installation and patches

The LispWorks 8.0 installer contains each of the Editions. Additionally, there may be a patch installer which upgrades LispWorks to version 8.0.x. You need to complete the main installation before adding patches.

3.2.2 Visual Studio runtime components and Windows Installer

On systems where this is not present, installing LispWorks will automatically install a copy of the Microsoft.VC80.CRT component, which contains the Microsoft Visual Studio runtime DLLs needed by LispWorks.

3.2.3 Installing over previous versions

You can install LispWorks 8.0 in the same location as LispWorks 7.1 or previous versions back to LispWorks 4.4.5. This is the default installation location.

You can also install LispWorks 8.0 without uninstalling older versions such as Xanalys LispWorks 4.4 or Xanalys LispWorks 4.3 provided that the chosen installation directory is different.

3.2.4 Information for Beta testers

Users of LispWorks 8.0 Beta should completely uninstall it before installing LispWorks 8.0. Remember to remove any patches added since the Beta release.

See **3.3 Uninstalling LispWorks for Windows** for instructions.

3.2.5 To install LispWorks

To install LispWorks (32-bit) for Windows run **`LispWorks80-32bit.exe`**. You will have downloaded this from the **`x86-win32`** folder.

To install LispWorks (64-bit) for Windows run **`LispWorks80-64bit.exe`**. You will have downloaded this from the **`x64-windows`** folder.

Follow the instructions on screen and read the remainder of this section.

3.2.5.1 Entering the License Data

Enter your serial number and license key when the installer asks for these details in the **Customer Information** screen.

Your license key will be supplied to you in email from Lisp Support or Lisp Sales.

If you have problems with your LispWorks license key, send it to **`lisp-keys@lispworks.com`**, describing what happens after you enter it, preferably with a screenshot.

Note: the LispWorks Personal Edition installer does not ask you to enter license data.

3.2.5.2 Installation location

By default 32-bit LispWorks installs in All Users space in **`C:\Program Files (x86)\LispWorks\`**.

By default 64-bit LispWorks installs in All Users space in **`C:\Program Files\LispWorks\`**.

To install LispWorks in a non-default location (for example, to ensure it is accessible only by the licensed user on a multi-user system such as a login server or remote desktop), select **Custom** setup in the **Setup Type** screen. Then click **Change...** in the **Custom Setup** screen and choose the desired location in the **Change Current Destination Folder** dialog. Do not simply move the LispWorks folder later, as this will break the installation.

3.2.5.3 Installing the Documentation

By default all the documentation is installed.

If you do not want to install the HTML Documentation, select **Custom** setup in the **Setup Type** screen and select **This feature will not be available** in the HTML Documentation feature in the **Custom Setup** screen.

You can also choose not to install the PDF Documentation, in a similar way.

You can add the HTML Documentation and the PDF Documentation later, by re-running the installer. The documentation is also available at **www.lispworks.com/documentation**.

3.2.5.4 Installing Patches

After completing the main installation of the Professional or Enterprise Edition, ensure you install the latest patches which are available for download at **www.lispworks.com/downloads/patch-selection.html**.

Patch installation instructions are in the README file accompanying the patch download.

3.2.5.5 Starting LispWorks

After installation LispWorks can be invoked from the Start menu or Start screen (on Windows 8).

Note: After installation you must not move or copy the LispWorks folder, since the system records the installation location. Moreover LispWorks needs to be able find its library at run time and therefore the LispWorks installation should not be moved around piecemeal. If you simply want to run LispWorks from somewhere more convenient, then consider adding a shortcut.

3.3 Uninstalling LispWorks for Windows

To uninstall LispWorks:

1. Select **Programs and Features** in the Control Panel or **App & features** in Settings on Windows 10.
2. Select **LispWorks 8.0 (32-bit)** or **LispWorks 8.0 (64-bit)** and click **Uninstall**.

This will uninstall LispWorks along with any installed updates. It will not remove any private patches.

3.4 Upgrading the LispWorks Edition

Some LispWorks features such as Delivery, Common SQL and KnowledgeWorks are not available in all Editions. You can add these features by upgrading.

After purchasing your upgrade from **lisp-sales@lispworks.com**, select **Help > Register...** and enter your new license key.

3.5 Upgrading to 64-bit LispWorks

To upgrade from 32-bit to 64-bit LispWorks, contact:

lisp-sales@lispworks.com

4 Installation on Linux

This chapter is an installation guide for LispWorks 8.0 (32-bit) for x86/x86_64 Linux, LispWorks 8.0 (64-bit) for x86_64 Linux, LispWorks 8.0 (32-bit) for ARM Linux and LispWorks 8.0 (64-bit) for ARM64 Linux. **10 Configuration on Linux, x86/x64 Solaris & FreeBSD** discusses post-installation and configuration in detail, but this chapter presents the instructions necessary to get LispWorks up and running on your system.

4.1 Software and hardware requirements

An overview of system requirements is provided in **System requirements on Linux**. The sections that follow discuss any relevant details.

System requirements on Linux

Hardware Requirements	Software Requirements
168MB of disk space for Enterprise Edition (32-bit) plus documentation	Any distribution with glibc 2.6 or later for x86/x86_64 and 2.17 or later for ARM/ARM64
182MB of disk space for Enterprise Edition (64-bit) plus documentation	GTK+ 2 (version 2.4 or higher) to run the GTK+ GUI. Open Motif 2.2.x or 2.3.x and Imlib2 1.4.3 or later to run the deprecated Motif GUI
Any modern machine is likely to have sufficient RAM to run LispWorks as distributed.	Firefox or Opera web browser for viewing on-line documentation

4.1.1 GUI libraries

LispWorks 8.0 for Linux requires that the X11 release 6 (or higher) is installed. It also requires that either GTK+ or Open Motif with Imlib2 are installed.

The remainder of this section contains the details for each of these distinct GUI options.

4.1.1.1 GTK+

In order for the LispWorks IDE to run "out of the box", GTK+ must be installed on the target machine.

GTK+ 2 (version 2.4 or higher) is required.

4.1.1.2 Motif

Open Motif version 2.2 or 2.3 is required to run LispWorks with the Motif GUI.

Download and install Open Motif 2.2.x or 2.3.x from your Linux distribution or from **www.motifzone.net**. Your systems administrator may be able to help if you do not know how to do this.

You will also need Imlib2 version 1.4.3 or later. Install this from your Linux distribution.

Note: You should be able to run the LispWorks 8.0 Motif GUI and LispWorks 7.x, LispWorks 6.x or LispWorks 5.x simultaneously with Open Motif installed.

4.1.2 Disk requirements

To install without documentation and optional modules, 32-bit LispWorks requires about 45MB and 64-bit LispWorks requires about 60MB. Installing the documentation adds about 110MB and the optional modules about 15MB. A full installation of the 64-bit Enterprise Edition with all documentation and optional modules requires about 185MB.

The documentation includes printable PDF format manuals. You may delete any of these that you do not need. They are available at **www.lispworks.com/documentation** in any case, and the same manuals are also available there in PostScript format.

4.2 License agreement

Before installing, you must read and agree to the license terms.

To do this download the license script from the link we sent to you.

Now run:

```
    sh lwl-license.sh
```

or, if you are installing the Personal Edition:

```
    sh lwlper-license.sh
```

Note: You must run this script as the same user that later performs the installation. In particular, if you are going to install LispWorks from the RPM file, you must run the license script while logged on as root.

Enter "yes" if you agree to the license terms.

4.3 Software delivery and installer formats

LispWorks 8.0 for Linux is supplied as a download. Two formats are provided:

- Red Hat Package Management (RPM) files for x86 and x86_64. RPM is a utility like `tar`, except it can actually install products after unpacking them. See **4.4.4 Installation from the binary RPM file (x86 and x86_64 only)** for more information.
- `tar` files.

4.3.1 Contents of the LispWorks distribution

The supplied installers contain all of the relevant modules.

For RPM installations, the RPM package name is `lispworks` (or `lispworks-personal` for the Personal Edition).

The Professional and Enterprise Edition modules are in separately installable RPM packages. These are: CLIM 2.0, KnowledgeWorks, LispWorks ORB, and Common SQL. **1.1 LispWorks Editions** provides Edition details.

For the Professional Edition the separately installable packages are:

```
    lispworks-clim
```

and for the Enterprise Edition the separately installable packages are:

```
lispworks-clim
lispworks-kw
lispworks-corba
lispworks-sql
```

The installation instructions provide the names of the individual distribution files.

4.4 Installing LispWorks for Linux

4.4.1 Main installation and patches

The LispWorks 8.0 installer contains each of the Editions. Additionally, there may be a patch installer which upgrades LispWorks to version 8.0.x. You need to complete the main installation before adding patches.

4.4.2 Installing over previous versions

You can install LispWorks 8.0 in the same location as LispWorks 7.1 or previous versions.

4.4.3 Information for Beta testers

Users of LispWorks 8.0 Beta should completely uninstall it (including any patches added to the beta installation) before installing LispWorks 8.0.

See **4.9 Uninstalling LispWorks for Linux** for instructions.

4.4.4 Installation from the binary RPM file (x86 and x86_64 only)

For installation on ARM and ARM64, see **4.4.5 Installation from the tar files**.

We recommend that you use RPM 4.3 or later (however see below for problems with **`--prefix`** argument with some versions of RPM). The distribution files are also provided in **`tar`** format in case you do not have a suitable version of RPM or are using another distribution of Linux.

If you already have LispWorks 8.0 Beta installed, please uninstall it before installing this product. See **4.9 Uninstalling LispWorks for Linux**.

Some versions of RPM may cause problems (eg. RPM 3.0). If you get the following message when using the **`--prefix`** argument:

```
rpm: only one of --prefix or --relocate may be used
```

try upgrading to RPM 3.0.2 or greater.

Installation of LispWorks for Linux from the RPM file must be done while you are logged on as root.

4.4.4.1 Installation directories

By default 32-bit LispWorks is installed in **`/usr/lib/LispWorks`** and a symbolic link to the executable is placed in **`/usr/bin/lispworks-8-0-0-x86-linux`**. Similarly, 64-bit LispWorks is installed in **`/usr/lib64/LispWorks`** and a symbolic link to the executable is placed in **`/usr/bin/lispworks-8-0-0-amd64-linux`**. However, the RPM is relocatable, and the **`--prefix`** option can be used to allow the installation of LispWorks in a non-default directory. The default prefix is **`/usr`**.

Note: RPM version 4.2 has a bug which can hinder secondary installations (CLIM, Common SQL, LispWorks ORB or KnowledgeWorks) in a user-specified directory. See **11.4.2 RPM_INSTALL_PREFIX not set** for a workaround.

Note: the Personal Edition installs by default in **`/usr/lib/LispWorksPersonal`**. Do not attempt to to install different editions in the same location, since some filenames coincide and uninstallation may break.

4.4.4.2 Selecting the correct RPM files

The main RPM file in the LispWorks distribution is named using the following pattern:

```
lispworks-8.0-n.arch.rpm
```

The integer *n* denotes a build number and will be same in all files in your distribution. The string *arch* will be either **`i386`** for 32-bit LispWorks or **`x86_64`** for 64-bit LispWorks. The text below assumes 32-bit LispWorks.

Note: For the Personal Edition, use **`lispworks-personal-8.0-*.i386.rpm`** wherever **`lispworks-8.0-*.i386.rpm`** is mentioned in this document. See **1.1.1 Personal Edition** for more information specific to the Personal Edition.

4.4.4.3 Installing or upgrading LispWorks for Linux

To install or upgrade LispWorks from the RPM file, perform the following steps as root:

1. Follow the instructions under **4.2 License agreement**.
2. Locate the RPM installation file **`lispworks-8.0-`***n***`.i386.rpm`**.
3. Install or upgrade LispWorks in the standard RPM way, for example:

   ```
   rpm --install lispworks-8.0-n.i386.rpm
   ```

   This command installs LispWorks in **`/usr/lib/LispWorks`**. A command line of the form:

   ```
   rpm --install --prefix <directory> lispworks-8.0-n.i386.rpm
   ```

   installs LispWorks in <*directory*>.

The directory name must be an absolute pathname. Relative pathnames and pathnames including shell-expanded characters such as **`.`** and **`~`** do not work.

Note: LispWorks needs to be able find its library at run time and therefore the LispWorks installation should not be moved around piecemeal. If you simply want to run LispWorks from somewhere more convenient, then consider adding a symbolic link.

See **4.6 Running LispWorks** for instructions on entering your license details.

4.4.4.4 Installing CLIM 2.0

The following module is packaged as a separate RPM file for installation after the main **`lispworks`** package. It is available in LispWorks Professional and Enterprise Editions only.

File distributions for layered products in Professional and Enterprise Editions

File Distribution	Layered Product
`lispworks-clim-8.0-`*`n`*`.i386.rpm`	CLIM 2.0

Install this module if required by substituting the above filename into the same commands you used to install the main **`lispworks`** package.

If you used a **`--prefix`** argument when installing LispWorks, then use the same prefix for this module.

4.4.4.5 Installing loadable Enterprise Edition modules

The following modules are packaged as separate RPM files for installation after the main **`lispworks`** package.

File distributions for layered products in the Enterprise Edition

File Distribution	Layered Product
`lispworks-clim-8.0-`*`n`*`.i386.rpm`	CLIM 2.0
`lispworks-kw-8.0-`*`n`*`.i386.rpm`	KnowledgeWorks
`lispworks-corba-8.0-`*`n`*`.i386.rpm`	LispWorks ORB
`lispworks-sql-8.0-`*`n`*`.i386.rpm`	Common SQL

Install these modules as described in **4.4.4.4 Installing CLIM 2.0**.

4.4.4.6 Documentation and saving space

Documentation in HTML and PDF format is provided with all editions. PostScript format is available to download. To obtain copies of the printable manuals, see **4.8 Printable LispWorks documentation**.

Documentation is installed by default in the **`lib/8-0-0-0/manual`** sub-directory of the LispWorks installation directory.

Using RPM, you can save space by choosing not to install the documentation. For example, use the following command (all on one line):

```
rpm --install --excludedocs --prefix <directory> lispworks-8.0-n.i386.rpm
```

To install the documentation at a later stage, you need to use the **`--replacepkgs`** option:

```
rpm --install --prefix <directory> --replacepkgs lispworks-8.0-n.i386.rpm
```

4.4.4.7 Installing Patches

After completing the main RPM installation of LispWorks and any modules, ensure you install the latest patches from the RPM file available for download at **www.lispworks.com/downloads/patch-selection.html**. Patch installation instructions are in the README file accompanying the patch download.

4.4.5 Installation from the tar files

The LispWorks distribution is also provided as **`tar`** files compressed using **`gzip`** for use if you do not have an appropriate version of RPM to unpack the RPM binary file. The gzipped files for LispWorks are as follows:

Files for LispWorks

`lw80-x86-linux.tar.gz`	32-bit LispWorks x86 image, modules and examples
`lw80-arm-linux.tar.gz`	32-bit LispWorks ARM image, modules and examples
`lw80-amd64-linux.tar.gz`	64-bit LispWorks x86_64 image, modules and examples
`lw80-arm64-linux.tar.gz`	64-bit LispWorks ARM64 image, modules and examples
`lwdoc80-x86-linux.tar.gz`	Documentation in HTML and PDF formats for all architectures

Note: The gzipped files for the LispWorks Personal Edition have similar names.

To install from these files:

1. Follow the instructions under **4.2 License agreement**.
2. Use **`cd`** to change directory to the location of the downloaded files before running the installation script.
3. Run the installation script **`lwl-install.sh`** (or **`lwlper-install.sh`** for the Personal Edition). as root if the directory specified by the installation directory requires it (the default does).

This script takes **`--prefix`** and **`--excludedocs`** arguments like **`rpm`** to control the installation directory and amount of documentation installed.

For example, to install the Personal Edition and documentation in the default location (**`/usr/local/lib/LispWorksPersonal`**) would use:

```
sh lwlper-install.sh
```

Or, to install 32-bit LispWorks in **`/usr/lispworks`**, without documentation you would use:

```
sh lwl-install.sh --excludedocs --prefix /usr/lispworks
```

Note: the default location under **`/usr/local`** is appropriate for this unmanaged (non-RPM) installation.

See **4.6 Running LispWorks** for how to enter your license details.

4.4.5.1 Installing Patches

After completing the main **`tar`** installation of LispWorks, ensure you install the latest patches from the **`tar`** archive available for download at **www.lispworks.com/downloads/patch-selection.html**. Patch installation instructions are in the README file accompanying the patch download.

4.5 LispWorks looks for a license key

If you try to run LispWorks without a valid key, it prints a message reporting that no valid key was found, and exits.

For instructions on entering your license key, see **4.6.1 Entering the license data** below.

For more information about license keys, see **10.2 License keys**.

4.6 Running LispWorks

In a RPM installation, assuming the default prefix of `/usr`, the LispWorks executable is located in `/usr/lib/LispWorks` or `/usr/lib64/LispWorks` or `/usr/lib/LispWorksPersonal` There is also a symbolic link from the `/usr/bin` directory.

In a `tar` installation, assuming the default prefix of `/usr/local`, the LispWorks executable is located in `/usr/local/lib/LispWorks` or `/usr/local/lib64/LispWorks` or `/usr/local/lib/LispWorksPersonal`.

In both cases, the LispWorks executable should not be moved without being resaved, because it needs to be able to locate the corresponding library directory on startup.

The LispWorks executable is named as shown here:.

`lispworks-personal-8-0-0-x86-linux`	Personal Edition
`lispworks-8-0-0-x86-linux`	32-bit LispWorks on x86
`lispworks-8-0-0-amd64-linux`	64-bit LispWorks on x86_64
`lispworks-8-0-0-arm-linux`	32-bit LispWorks on ARM
`lispworks-8-0-0-arm64-linux`	64-bit LispWorks on ARM64

When you run LispWorks, the splashscreen should appear, followed by the LispWorks Podium and a Listener. See **11.1 Troubleshooting** for details if this does not happen.

4.6.1 Entering the license data

When you run LispWorks for the first time, you will need to enter your license details. This should be done as follows (all on one line) using the appropriate LispWorks executable from the table above (32-bit LispWorks on x86 in this example):

```
lispworks-8-0-0-x86-linux --lwlicenseserial SERIALNUMBER --lwlicensekey LICENSEKEY
```

where *SERIALNUMBER* and *LICENSEKEY* are the strings supplied with LispWorks. A message:

```
LispWorks license installed successfully.
```

should be printed and thereafter you can run LispWorks without those command line arguments.

Your license key will be supplied to you in email from Lisp Support or Lisp Sales.

If you have problems with your LispWorks license key, send it to `lisp-keys@lispworks.com`, showing the complete output after you enter it.

Note: the LispWorks Personal Edition does not ask you to enter license data.

4.7 Configuring the image

You can now configure your LispWorks image to suit your needs and load modules as necessary. For instructions, see **10 Configuration on Linux, x86/x64 Solaris & FreeBSD**.

4.8 Printable LispWorks documentation

In a default installation, the **`lib/8-0-0-0/manual/offline`** directory contains PDF format versions of the manuals.

These files are also available from **www.lispworks.com/documentation**.

PostScript format versions of the manuals are also available for download.

4.9 Uninstalling LispWorks for Linux

A RPM installation of LispWorks can be uninstalled in the usual way, for example by executing this command, as root:

```
rpm --erase lispworks-8.0
```

If patches have been added via RPM, then you will first need to uninstall that package, which will be named **`lispworks-patches8.0`**. The same applies to additional RPM packages such as **`lispworks-sql`**.

If patches have been added from a **`tar`** archive, you will need to remove them by hand.

If you installed LispWorks from the **`tar`** archives, simply do:

```
rm -rf /usr/local/lib/LispWorks
```

4.10 Upgrading the LispWorks Edition

Some LispWorks features such as Delivery, Common SQL and KnowledgeWorks are not available in all Editions. You can add these features by upgrading.

After purchasing your upgrade from **lisp-sales@lispworks.com**, select **Help > Register...** and enter your new license key.

4.11 Upgrading to 64-bit LispWorks

To upgrade from 32-bit to 64-bit LispWorks, contact:

lisp-sales@lispworks.com

5 Installation on x86/x64 Solaris

This chapter is an installation guide for LispWorks 8.0 (32-bit) for x86/x64 Solaris and LispWorks 8.0 (64-bit) for x86/x64 Solaris. **10 Configuration on Linux, x86/x64 Solaris & FreeBSD** discusses post-installation and configuration in detail, but this chapter presents the instructions necessary to get LispWorks up and running on your system.

5.1 Software and hardware requirements

An overview of system requirements is provided in **System requirements on x86/x64 Solaris**. The sections that follow discuss any relevant details.

System requirements on x86/x64 Solaris

Hardware Requirements	Software Requirements
For 32-bit LispWorks, 157MB of disk space	Solaris 10 (release 5/08 or later), Solaris 11, or OpenSolaris (release 2009.06 or later)
For 64-bit LispWorks, 171MB of disk space	GTK+ 2 (version 2.4 or higher) to run the GTK+ GUI. Motif 2.1 and Imlib to run the deprecated Motif GUI
Any modern machine is likely to have sufficient RAM to run LispWorks as distributed.	Firefox or Opera web browser for viewing on-line documentation

5.1.1 GUI libraries

LispWorks 8.0 for x86/x64 Solaris requires that the X11 release 6 (or higher) is installed. It also requires that either GTK+ or Motif with Imlib are installed.

The remainder of this section contains the details for each of these distinct GUI options.

5.1.1.1 GTK+

In order for the LispWorks IDE to run "out of the box", GTK+ must be installed on the target machine.

GTK+ 2 (version 2.4 or higher) is required.

5.1.1.2 Motif

Motif 2.1 or higher is required to run LispWorks with the Motif GUI.

The Motif libraries are installed as part of the SUNWmfrun package. It is usually preinstalled on Solaris 10 and is available for download from Sun for OpenSolaris.

You will also need Imlib (not Imlib2). Imlib version 1.9.13 or later is recommended. Contact Lisp Support if you need this.

5.1.2 Disk requirements

32-bit LispWorks requires about 130MB to install.

64-bit LispWorks requires about 140MB to install.

The installation includes about 70MB of documentation.

The documentation includes printable PDF format manuals. You may delete any of these that you do not need. They are available at **www.lispworks.com/documentation** in any case, and the same manuals are also available there in PostScript format.

5.2 Software delivery and installer format

LispWorks 8.0 for x86/x64 Solaris is supplied as a standard package file to download.

There are two variants, 32-bit LispWorks and 64-bit LispWorks, so be sure to download the one for which you have purchased a license:

5.2.1 Contents of the LispWorks distribution

All of the LispWorks modules are contained in a single package file. Your license key will control which modules can be used.

The package name for 32-bit LispWorks is **`LispWorks80-32bit`**.

The package name for 64-bit LispWorks is **`LispWorks80-64bit`**.

5.2.2 Personal Edition distribution

You can install the LispWorks Personal Edition by downloading it from **www.lispworks.com/downloads**.

The package for the Personal Edition is **`LispWorksPersonal80-32bit`**.

5.3 Installing LispWorks for x86/x64 Solaris

5.3.1 Main installation and patches

The LispWorks 8.0 installer contains each of the Editions. Additionally, there may be a patch installer which upgrades LispWorks to version 8.0.x. You need to complete the main installation before adding patches.

5.3.2 Installing over previous versions

You can install LispWorks 8.0 in the same location as LispWorks 7.1 or previous versions.

5.3.3 Information for Beta testers

Users of LispWorks 8.0 Beta should completely uninstall it (including any patches added to the beta installation) before installing LispWorks 8.0.

See **5.8 Uninstalling LispWorks for x86/x64 Solaris** for instructions.

5.3.4 Installation directories

32-bit LispWorks is installed by default in **`/opt/LispWorks/lib/LispWorks`** and a symbolic link to the executable is placed in **`/opt/LispWorks/bin/lispworks-8-0-0-x86-solaris`**.

64-bit LispWorks is installed by default in **`/opt/LispWorks/lib/amd64/LispWorks`** and a symbolic link to the executable is placed in **`/opt/LispWorks/bin/lispworks-8-0-0-amd64-solaris`**.

LispWorks Personal Edition is installed by default in **`/opt/LispWorks/lib/LispWorksPersonal`** and a symbolic link to the executable is placed in **`/opt/LispWorks/bin/lispworks-personal-8-0-0-x86-solaris`**.

Note: LispWorks needs to be able find its library at run time and therefore the LispWorks installation should not be moved around piecemeal. If you simply want to run LispWorks from somewhere more convenient, then consider adding a symbolic link.

5.3.5 Selecting the correct software package file

The 32-bit LispWorks software package file is called **`LispWorks80-32bit`**.

The 64-bit LispWorks software package file is called **`LispWorks80-64bit`**.

The Personal Edition software package file is called **`LispWorksPersonal80-32bit`**.

Note: the software may be supplied in a compressed format with a **`.gz`** extension. Uncompress it using **`gunzip`**.

5.3.6 Installing the package file

To install LispWorks, perform the following steps as root:

1. Locate the software package file.
2. Install or upgrade LispWorks in the standard way, for example:

   ```
   pkgadd -d LispWorks80-32bit all
   ```

   for 32-bit LispWorks, or:

   ```
   pkgadd -d LispWorks80-64bit all
   ```

   for 64-bit LispWorks.
3. The license terms are presented. Enter "yes" if you agree to them.

See **5.5 Running LispWorks** for instructions on entering your license serial number and key.

5.3.7 Installing Patches

After completing the main installation of LispWorks, ensure you install the latest patches from the package file available for download at **www.lispworks.com/downloads/patch-selection.html**. Patch installation instructions are in the README file accompanying the patch download.

5.4 LispWorks looks for a license key

If you try to run LispWorks without a valid key, it prints a message reporting that no valid key was found, and exits.

For instructions on entering your license key, see **5.5.1 Entering the license data** below.

For more information about license keys, see **10.2 License keys**.

5.5 Running LispWorks

Run LispWorks (all variants) from the directory **`/opt/LispWorks/bin`**.

The LispWorks executable is named as shown here:

`lispworks-personal-8-0-0-x86-solaris`	Personal Edition
`lispworks-8-0-0-x86-solaris`	32-bit LispWorks
`lispworks-8-0-0-amd64-solaris`	64-bit LispWorks

This executable should not be moved without being resaved because it needs to be able to locate the corresponding library directory on startup.

When you run LispWorks, the splashscreen should appear, followed by the LispWorks Podium and a Listener. See **11.1 Troubleshooting** for details if this does not happen.

5.5.1 Entering the license data

When you run LispWorks for the first time, you will need to enter your license details. This should be done as follows (all on one line):

```
lispworks-8-0-0-x86-solaris --lwlicenseserial SERIALNUMBER --lwlicensekey LICENSEKEY
```

where *SERIALNUMBER* and *LICENSEKEY* are the strings supplied with LispWorks. A message:

```
LispWorks license installed successfully.
```

should be printed and thereafter you can run LispWorks without those command line arguments.

Your license key will be supplied to you in email from Lisp Support or Lisp Sales.

If you have problems with your LispWorks license key, send it to **`lisp-keys@lispworks.com`**, showing the complete output after you enter it.

Note: the LispWorks Personal Edition does not ask you to enter license data.

5.6 Configuring the image

You can now configure your LispWorks image to suit your needs and load modules as necessary. For instructions, see **10 Configuration on Linux, x86/x64 Solaris & FreeBSD**.

5.7 Printable LispWorks documentation

In a default installation, the **`lib/8-0-0-0/manual/offline`** directory contains PDF format versions of the manuals.

These files are also available at **www.lispworks.com/documentation/**.

PostScript format versions of the manuals are also available for download.

5.8 Uninstalling LispWorks for x86/x64 Solaris

To uninstall LispWorks, perform the following steps as root:

1. If patches for LispWorks 8.0 have been installed then you will need to uninstall the patch package, by:

   ```
   pkgrm -n LispWorksPatches80-32bit
   ```

   or:

   ```
   pkgrm -n LispWorksPatches80-64bit
   ```

2. Then uninstall the main software package containing LispWorks 8.0 by executing:

   ```
   pkgrm -n LispWorks80-32bit
   ```

   or:

   ```
   pkgrm -n LispWorks80-64bit
   ```

5.9 Upgrading the LispWorks Edition

Some LispWorks features such as Delivery, Common SQL and KnowledgeWorks are not available in all Editions. You can add these features by upgrading.

After purchasing your upgrade from **lisp-sales@lispworks.com**, select **Help > Register...** and enter your new license key.

5.10 Upgrading to 64-bit LispWorks

To upgrade from 32-bit to 64-bit LispWorks, contact:

lisp-sales@lispworks.com

6 Installation on FreeBSD

This chapter is an installation guide for LispWorks 8.0 (32-bit) for FreeBSD and LispWorks 8.0 (64-bit) for FreeBSD. **10 Configuration on Linux, x86/x64 Solaris & FreeBSD** discusses post-installation and configuration in detail, but this chapter presents the instructions necessary to get LispWorks up and running on your system.

6.1 Software and hardware requirements

An overview of system requirements is provided in **System requirements on FreeBSD**. The sections that follow discuss any relevant details.

System requirements on FreeBSD

Hardware Requirements	Software Requirements
168MB of disk space for 32-bit LispWorks plus documentation	FreeBSD 10.x, or later with compat10x (if you want to run LispWorks on older versions of FreeBSD, then please contact Lisp Support)
182MB of disk space for 64-bit LispWorks plus documentation	GTK+ 2 (version 2.4 or higher) to run the GTK+ GUI. Open Motif 2.3.x and Imlib2 1.4.9 or later to run the deprecated Motif GUI
Any modern machine is likely to have sufficient RAM to run LispWorks as distributed.	Firefox or Opera web browser for viewing on-line documentation

6.1.1 GUI libraries

LispWorks 8.0 for FreeBSD requires that the X11 release 6 (or higher) is installed.

LispWorks 8.0 also requires that either GTK+ or Open Motif with Imlib2 are installed.

The remainder of this section contains the details for each of these distinct GUI options.

6.1.1.1 GTK+

In order for the LispWorks IDE to run "out of the box", GTK+ must be installed on the target machine.

GTK+ 2 (version 2.4 or higher) is required.

6.1.1.2 Motif

Open Motif version 2.3 is required to run LispWorks with the Motif GUI.

Install Open Motif 2.3.x from the FreeBSD distribution or ports tree. Your systems administrator may be able to help if you do not know how to do this.

You will also need Imlib2 version 1.4.9 or later. Install this from the FreeBSD distribution or ports tree.

6.1.2 Disk requirements

32-bit LispWorks requires about 160MB to install, and 64-bit LispWorks needs 180MB. This includes 110MB of documentation.

The documentation includes printable PDF format manuals. You may delete any of these that you do not need. They are available at **www.lispworks.com/documentation** in any case, and the same manuals are also available there in PostScript format.

6.2 License agreement

Before installing, you must read and agree to the license terms.

To do this download the license script from the link we sent to you.

Now run:

```
sh lwf-license.sh
```

or, if you are installing the Personal Edition:

```
sh lwfper-license.sh
```

Note: You must run this script as the same user that later performs the installation.

Enter "yes" if you agree to the license terms.

6.3 Software delivery and installer format

LispWorks 8.0 for FreeBSD is supplied as a standard package file, in pkg(8) format, to download.

6.3.1 Contents of the LispWorks distribution

All of the LispWorks modules are contained in a single package file. Your license key will control which modules can be used.

The package name for 32-bit LispWorks is **lispworks80-32bit**.

The package name for 64-bit LispWorks is **lispworks80-64bit**.

6.3.2 Personal Edition distribution

You can install the LispWorks Personal Edition by downloading it from **www.lispworks.com/downloads**.

The package name for the Personal Edition is **lispworks80-personal**.

6.4 Installing LispWorks for FreeBSD

6.4.1 Main installation and patches

The LispWorks 8.0 installer contains each of the Editions. Additionally, there may be a patch installer which upgrades LispWorks to version 8.0.x. You need to complete the main installation before adding patches.

6.4.2 Installing over previous versions

You can install LispWorks 8.0 in the same location as LispWorks 7.1 or previous versions.

6.4.3 Information for Beta testers

Users of LispWorks 8.0 Beta should completely uninstall it (including any patches added to the beta installation) before installing LispWorks 8.0.

See **6.9 Uninstalling LispWorks for FreeBSD** for instructions.

6.4.4 Installation directories

By default LispWorks is installed in **`/usr/local/lib/LispWorks`**. A symbolic link to the 32-bit executable is placed in **`/usr/local/bin/lispworks-8-0-0-x86-freebsd`**. A symbolic link to the 64-bit executable is placed in **`/usr/bin/lispworks-8-0-0-amd64-freebsd`**.

Note: the Personal Edition by default installs in **`/usr/local/lib/LispWorksPersonal`**. Do not attempt to to install different editions in the same location, since some filenames coincide and uninstallation may break.

6.4.5 Selecting the correct software package file

The 32-bit LispWorks software package file is called:

```
lispworks80-32bit-8.0.txz
```

The 64-bit LispWorks software package file is called:

```
lispworks80-64bit-8.0.txz
```

The Personal Edition software package file is called:

```
lispworks80-personal-8.0.txz
```

6.4.6 Installing LispWorks for FreeBSD

To install LispWorks, perform the following steps as root:

1. Follow the instructions under **6.2 License agreement**.
2. Locate the software package file.
3. Install or upgrade LispWorks in the standard way, for example:

   ```
   pkg add lispworks80-32bit-8.0.txz
   ```

   This command installs LispWorks in **`/usr/local/lib/LispWorks`**.

Note: LispWorks needs to be able find its library at run time and therefore the LispWorks installation should not be moved around piecemeal. If you simply want to run LispWorks from somewhere more convenient, then consider adding a symbolic link.

See **6.6 Running LispWorks** for instructions on entering your license details.

6.4.7 Installing Patches

After completing the main installation of LispWorks, ensure you install the latest patches from the package file available for download at **www.lispworks.com/downloads/patch-selection.html**. Patch installation instructions are in the README file accompanying the patch download.

6.5 LispWorks looks for a license key

If you try to run LispWorks without a valid key, it prints a message reporting that no valid key was found, and exits.

For instructions on entering your license key, see **6.6.1 Entering the license data** below.

For more information about license keys, see **10.2 License keys**.

6.6 Running LispWorks

The LispWorks executable is located in the **`/usr/local/lib/LispWorks`** or **`/usr/local/lib/LispWorksPersonal`** directory of the installation (assuming the default prefix of **`/usr/local`**) and should not be moved without being resaved because it needs to be able to locate the corresponding library directory on startup. There is also a symbolic link from the **`/usr/local/bin`** directory.

The LispWorks executable is named as shown here:.

`lispworks-personal-8-0-0-x86-freebsd`	Personal Edition
`lispworks-8-0-0-x86-freebsd`	32-bit LispWorks
`lispworks-8-0-0-amd64-freebsd`	64-bit LispWorks

When you run LispWorks, the splashscreen should appear, followed by the LispWorks Podium and a Listener. See **11.1 Troubleshooting** for details if this does not happen.

6.6.1 Entering the license data

When you run LispWorks for the first time, you will need to enter your license details. This should be done as follows (all on one line):

```
lispworks-8-0-0-x86-freebsd --lwlicenseserial SERIALNUMBER --lwlicensekey LICENSEKEY
```

where *SERIALNUMBER* and *LICENSEKEY* are the strings supplied with LispWorks. A message:

```
LispWorks license installed successfully.
```

should be printed and thereafter you can run LispWorks without those command line arguments.

Your license key will be supplied to you in email from Lisp Support or Lisp Sales.

If you have problems with your LispWorks license key, send it to **`lisp-keys@lispworks.com`**, showing the complete output after you enter it.

Note: the LispWorks Personal Edition does not ask you to enter license data.

6.7 Configuring the image

You can now configure your LispWorks image to suit your needs and load modules as necessary. For instructions, see **10 Configuration on Linux, x86/x64 Solaris & FreeBSD**.

6.8 Printable LispWorks documentation

In a default installation, the **`lib/8-0-0-0/manual/offline`** directory contains PDF format versions of the manuals.

These files are also available at **www.lispworks.com/documentation/**.

PostScript format versions of the manuals are also available for download.

6.9 Uninstalling LispWorks for FreeBSD

To uninstall LispWorks, perform the following steps as root:

1. If patches have been installed, then you will first need to uninstall that package:

   ```
   pkg delete lispworks80-patches-32bit
   ```

   or:

   ```
   pkg delete lispworks80-patches-64bit
   ```

2. Then uninstall the main software package containing LispWorks 8.0:

   ```
   pkg delete lispworks80-32bit
   ```

   or:

   ```
   pkg delete lispworks80-64bit
   ```

6.10 Upgrading the LispWorks Edition

Some LispWorks features such as Delivery, Common SQL and KnowledgeWorks are not available in all Editions. You can add these features by upgrading.

After purchasing your upgrade from **lisp-sales@lispworks.com**, select **Help > Register...** and enter your new license key.

6.11 Upgrading to 64-bit LispWorks

To upgrade from 32-bit to 64-bit LispWorks, contact:

> **lisp-sales@lispworks.com**

7 Installation of LispWorks for Mobile Runtime

This chapter describes installation of LispWorks 8.0 for Android Runtime and LispWorks 8.0 for iOS Runtime.

7.1 Installing LispWorks for Android Runtime

We will send you instructions when you get a license for LispWorks for Android Runtime.

Note: Normally you would first develop and debug your program using LispWorks on a desktop platform, for example LispWorks for Linux. You will then build a runtime library using LispWorks for Android Runtime and incorporate it in an Android project (see "Android interface" in the *LispWorks® User Guide and Reference Manual*) before testing it on an Android device.

7.2 Installing LispWorks for iOS Runtime

We will send you instructions when you get a license for LispWorks for iOS Runtime.

Note: Normally you would first develop and debug your program using LispWorks for Macintosh. You will then build a runtime library using LispWorks for iOS Runtime and incorporate it in an Xcode project (see "iOS interface" in the *LispWorks® User Guide and Reference Manual*) before testing it on an iOS device or the iOS Simulator on macOS.

8 Configuration on macOS

8.1 Introduction

This chapter explains how to get LispWorks up and running, having already installed the files into an appropriate folder. If you have not done this, refer to **2 Installation on macOS**.

It is more useful to have an image customized to suit your particular environment and work needs. You can do this—setting useful pathnames, loading libraries, and so on—and then save the image to create another that will be configured as you require whenever you start it up.

This chapter covers the following topics:

- **8.2 License keys**
- **8.3 Configuring your LispWorks installation**
- **8.4 Saving and testing the configured image**
- **8.5 Initializing LispWorks**
- **8.6 Loading CLIM 2.0**
- **8.7.1 Loading Common SQL**
- **8.8 Common Prolog and KnowledgeWorks**

8.2 License keys

LispWorks is protected against unauthorized copying and use by a simple key mechanism. LispWorks will not start up until it finds a file containing a valid key.

The image looks for a file `lwlicense` in the following places, in order:

- In the current working directory (folder).
- In the directory containing the LispWorks executable.
- In the `Library/lib/8-0-0-0/config` subdirectory of the LispWorks installation directory.

When the file `lwlicense` is found, it must contain a valid key for the current machine. If you try to run LispWorks without a valid key, a message will be printed to the console reporting that no valid key was found, and LispWorks will exit.

8.3 Configuring your LispWorks installation

Once you have successfully installed and run LispWorks, you can configure it to suit your local conditions and needs, producing an image that is set up the way you want it to be every time you start it up.

8.3.1 Levels of configuration

There are two levels of configuration:

- Configuring and resaving the image, thereby creating a new image that is exactly as you want it at startup.
- Configuring certain aspects of LispWorks as it starts up.

These two levels are available for good reason: while some configuration details may be of use to all LispWorks users on your machine (for instance, having a particular library built into the image where before it was only load-on-demand) others may be a matter of personal preference (for instance how many editor windows are allowed on-screen, or the colors of tool windows).

In the first case, you use edited copies of files in the **`config`** folder to achieve your aims.

In the second case, you make entries in your initialization file. This is a file read every time LispWorks starts up, and it can contain any valid Common Lisp code. (Most of the configurable settings in LispWorks can be controlled from Common Lisp.) By default the file is called **`.lispworks`** and is in your home directory. Your initialization file can be changed via **LispWorks > Preferences...** from the LispWorks IDE.

8.3.2 Configuring images for the different GUIs

If you have installed both the LispWorks images, for native macOS and for GTK+, you will want to configure two images.

If necessary your Lisp configuration and initialization files can run code for one image or the other by conditionalization on the feature **`:cocoa`**. The native macOS LispWorks image has **`:cocoa`** on **`*features*`** while the GTK+ LispWorks image does not, and has **`:gtk`**.

8.3.3 Configuration files available

There are four sample configuration files in LispWorks library containing settings you can change in order to configure images:

- **`config/configure.lisp`**
- **`config/siteinit.lisp`**
- **`private-patches/load.lisp`**
- **`config/a-dot-lispworks.lisp`**

`config/configure.lisp` is preloaded into the image before it is shipped. It contains settings governing fundamental issues like where to find the LispWorks run time folder structure, and so on. You can override these settings in your saved image or in your initialization file. You should read through **`configure.lisp`** .

`config/siteinit.lisp` contains any forms that are appropriate to the whole site but which are to be loaded afresh each time the image is started. The sample **`siteinit.lisp`** file distributed with LispWorks contains only the form:

```
(load-all-patches)
```

On startup, the image loads **`siteinit.lisp`** and your initialization file, in that order. The command line options **`-siteinit`** and **`-init`** can be used to specify loading of different files or to suppress them altogether. See the example in **8.4 Saving and testing the configured image**, below, and **8.5 Initializing LispWorks** for further details.

`private-patches/load.lisp` is loaded by **`load-all-patches`**, and should contain forms to load any private (named) patches that Lisp Support might send you.

`config/a-dot-lispworks.lisp` is a sample personal initialization file. You might like to copy this into a file

`~/.lispworks` in your home directory and edit it to create your own initialization file.

Both **`configure.lisp`** and **`a-dot-lispworks.lisp`** are preloaded into the image before it is shipped, so if you are happy with the settings in these files, you need not change them. See the example in **8.4 Saving and testing the configured image**, below, and **8.5 Initializing LispWorks** for further details.

8.4 Saving and testing the configured image

It is not usually necessary to save an image merely to preload patches and your configuration, because these load very quickly on modern machines.

However, if you want to save an image to reduce startup time for a complex configuration (such as large application code) or to save a non-windowing image, then proceed as described in this section.

8.4.1 Create a configuration file

Make a copy of **`config/configure.lisp`** called **`/tmp/my-configuration.lisp`**. When you have made the desired changes in **`my-configuration.lisp`** you can save a new LispWorks image as described in **8.4.2 Create and use a save-image script**.

8.4.2 Create and use a save-image script

1. Create a configuration and saving script **`/tmp/save-config.lisp`** containing:

   ```
   (in-package "CL-USER")
   (load-all-patches)
   (load "/tmp/my-configuration.lisp")
   #+:cocoa
   (save-image-with-bundle "/Applications/My LispWorks/LW")
   #-:cocoa
   (save-image "my-lispworks-gtk")
   ```

2. Change directory to the directory containing the LispWorks image to configure. For the native macOS/Cocoa LispWorks image:

   ```
   % cd "/Applications/LispWorks 8.0 (64-bit)/LispWorks (64-bit).app/Contents/MacOS"
   ```

   or for the X11/GTK+ LispWorks image:

   ```
   % cd "/Applications/LispWorks 8.0 (64-bit)"
   ```

3. Start the supplied image passing the configuration script the build file. For example enter one of the following commands (on one line of input):

   ```
   % ./lispworks-8-0-0-macos64-universal -build /tmp/save-config.lisp
   ```

   or:

   ```
   % ./lispworks-8-0-0-macos64-universal-gtk -build /tmp/save-config.lisp
   ```

   If the image will not run at this stage, it is probably not finding a valid key.

   Saving the image takes some time.

You can now use the new **`My LispWorks/LW.app`** application bundle or the **`my-lispworks-gtk`** image by starting it just as you did the supplied LispWorks. The supplied LispWorks is not required after the configuration process has been successfully completed.

Do not try to save a new image over an image that is currently running. Instead, save an image under a unique name, and then, if necessary, replace the new image with the old one after the call to **`save-image`** has returned.

8.4.3 What to do if no image is saved

If no new image is saved, then there is some error while loading the build script. To see the error message, run the command with output redirected to a file, for example:

```
% ./lispworks-8-0-0-macos64-universal -build /tmp/save-config.lisp > /tmp/output.txt
```

Look in the file **`/tmp/output.txt`**.

8.4.4 Testing the newly saved image

You should now test the new LispWorks image. To test a configured LispWorks, do the following:

1. If you are using an X11/GTK+ image, change directory to **`/tmp`**.
2. When using X11, verify that your **`DISPLAY`** environment variable is correctly set and that your machine has permission to connect to the display.
3. Start up the new image, by entering the path of the X11/GTK+ executable or by double-clicking on the LispWorks icon in the macOS Finder.

   The window-based environment should now initialize—during initialization a window displaying a copyright notice will appear on the screen.

   You may wish to work through some of the examples in the *LispWorks® User Guide and Reference Manual*, to further check that the configured image has been successfully built.
4. Test the load-on-demand system. In the Listener, type:

   ```
   CL-USER 1 > (inspect 1)
   ```

   Before information about the fixnum 1 is printed, the system should load the inspector from the **`load-on-demand`** Library directory.

   You can quit the inspector by typing **`:q`**.

8.4.5 Saving a non-windowing image

For some purposes such as scripting it is convenient to have a LispWorks image that does not start the graphical programming environment.

To save an image which does not automatically start the GUI, use a script as described in **8.4.2 Create and use a save-image script** but pass the **`:environment`** argument to **`save-image`**. For example:

```
(save-image "my-tty-lispworks" :environment nil)
```

8.5 Initializing LispWorks

When LispWorks starts up, it looks for an initialization file to load. The name of the file is held in `*init-file-name*`, and is `~/.lispworks` by default. The '`~`' denotes your home directory, indicated as **Home** in the Finder. The initialization file may contain any valid Lisp code.

You can load a different initialization file using the option `-init` in the command line, for example:

```
% "/Applications/LispWorks 8.0 (64-bit)/LispWorks (64-bit).app/Contents/MacOS/lispworks-8-0-0-
macos64-universal" -init my-lisp-init
```

(where `%` denotes the Unix shell prompt) would make LispWorks load `my-lisp-init.lisp` as the initialization file instead of that named by `*init-file-name*`.

The loading of the siteinit file (located by default at `config/siteinit.lisp`) is similarly controlled by the `-siteinit` command line argument or `*site-init-file-name*`.

You can start an image without loading any personal or site initialization file by passing a hyphen to the `-init` and `-siteinit` arguments instead of a filename:

```
% "/Applications/LispWorks 8.0 (64-bit)/LispWorks (64-bit).app/Contents/MacOS/lispworks-8-0-0-
macos64-universal" -init - -siteinit -
```

This starts the LispWorks image without loading any initialization file. It is often useful to start the image in this way when trying to repeat a suspected bug. You should always start the image without the default initialization files if you are intending to resave it.

In all cases, if the filename is present, and is not a hyphen, LispWorks tries to load it as a normal file by calling `load`. If the load fails, LispWorks prints an error report.

8.6 Loading CLIM 2.0

CLIM 2.0 is supported on the X11/Motif GUI.

Load CLIM 2.0 into the "LispWorks for X11 IDE" image with:

```
(require "clim")
```

and the CLIM demos with:

```
(require "clim-demo")
```

A configuration file to save an image with CLIM 2.0 preloaded would look something like this:

```
(in-package "CL-USER")
(load-all-patches)
(require "clim")
(save-image "/path/to/clim-lispworks")
```

To run the demo software, enter the following in a listener:

```
(require "clim-demo")
(clim-demo:start-demo)
```

Note: CLIM is not supported by the LispWorks native macOS image and cannot be loaded into it.

Note: CLIM is not supported under GTK+.

Note: Do not attempt to load CLIM via the clim loader files in the clim distribution. This will cause CLIM patches to not be loaded. Use **`(require "clim")`**.

8.7 The Common SQL interface

The Common SQL interface requires ODBC or one of the supported database types listed in section "Supported Databases" of the *LispWorks® User Guide and Reference Manual*.

8.7.1 Loading Common SQL

To load Common SQL enter, for example:

```
(require "odbc")
```

or:

```
(require "oracle")
```

Initialize the database type at run time, for example:

```
(sql:initialize-database-type :database-type :odbc)
```

or:

```
(sql:initialize-database-type :database-type :oracle)
```

See the *LispWorks® User Guide and Reference Manual* for further information.

8.7.2 Supported databases

Common SQL on macOS has been tested with DBMS Postgres 7.2.1, MySQL 5.0.18, Oracle Instant Client 10.2.0.4, ODBC driver PSQLODBC development code, and IODBC as supplied with macOS.

8.7.3 Special considerations when using Common SQL

8.7.3.1 Location of .odbc.ini

The current release of macOS comes with an ODBC driver manager from IODBC, including a GUI interface. IODBC attempts to put the file **`.odbc.ini`** file in a non-standard location. This causes problems at least with the PSQLODBC driver for PostgreSQL, because PSQLODBC expects to find **`.odbc.ini`** in either the users's home directory or the current directory. There may be similar problems with other drivers. Therefore the file **`.odbc.ini`** should be placed in its standard place **`~/.odbc.ini`**. The IODBC driver manager looks there too, so it will work.

8.7.3.2 Errors using PSQLODBC

The PSQLODBC driver, when it does not find any of the Servername, Database or Username in **`.odbc.ini`**, returns the wrong error code. This tells the calling function that the user cancelled the login dialog.

Therefore, if Common SQL reports that the user cancelled when trying to connect, you need to check that you have got

Servername, Database and Username, with the correct case, in the section for the datasource in the **`.odbc.ini`** file.

Note: Username may alternatively be given in the connect string.

8.7.3.3 psqlODBC version

Common SQL was tested with the development version of psqlODBC (that is downloaded from CVS), with the version changed to 3. Contact Lisp Support if you need help using Common SQL with psqlODBC.

8.7.3.4 Locating the Oracle, MySQL or PostgreSQL client libraries

For *database-type* **`:oracle`**, **`:mysql`** and **`:postgresql`**, if the client library is not installed in a standard place, its directory must be added to the environment variable DYLD_LIBRARY_PATH (see the OS manual entry for dyld).

8.8 Common Prolog and KnowledgeWorks

Common Prolog is bundled with KnowledgeWorks rather than with LispWorks. KnowledgeWorks is loaded by using:

```
(require "kw")
```

See the *KnowledgeWorks and Prolog User Guide* for further instructions.

9 Configuration on Windows

9.1 Introduction

This chapter explains how to get LispWorks up and running, having already installed it If you have not done this, refer to **3 Installation on Windows**.

It is more useful to have an image customized to suit your particular environment and work needs. You can do this—setting useful pathnames, loading libraries, and so on—and then save the image to create another that will be configured as you require whenever you start it up.

This chapter covers the following topics:

- **9.2 License keys**
- **9.3 Configuring your LispWorks installation**
- **9.4 Saving and testing the configured image**
- **9.5 Initializing LispWorks**
- **9.6 Loading CLIM 2.0**
- **9.7 The Common SQL interface**
- **9.8 Common Prolog and KnowledgeWorks**

9.2 License keys

LispWorks is protected against unauthorized copying and use by a simple key protection mechanism. LispWorks will not start up until it finds a valid key.

The image looks for a valid license key in the Windows registry.

If you try to run LispWorks without a valid key, it will prompt for a serial number and key.

9.3 Configuring your LispWorks installation

Once you have successfully installed and run LispWorks, you can configure it to suit your local conditions and needs, producing an image that is set up the way you want it to be every time you start it up.

9.3.1 Levels of configuration

There are two levels of configuration: configuring and resaving the image, thereby creating a new image that is exactly as you want it at startup, and configuring certain aspects of LispWorks as it starts up.

These two levels are available for good reason: while some configuration details may be of use to all LispWorks users on your site (for instance, having a particular library built in to the image where before it was only load-on-demand) others may be a matter of personal preference (for instance how many editor windows are allowed on-screen, or the colors of tool windows).

In the first case, you use edited copies of files in the **`config`** folder to achieve your aims.

In the second case, you make entries in your initialization file. This is a file read every time LispWorks starts up, and it can contain any valid Common Lisp code. (Most of the configurable settings in LispWorks can be controlled from Common Lisp.) Your initialization file can be changed via **`Tools > Preferences...`** in the LispWorks IDE.

9.3.2 Configuration files available

There are four sample configuration files in LispWorks library containing settings you can change in order to configure images:

- **`config/configure.lisp`**
- **`config/siteinit.lisp`**
- **`private-patches/load.lisp`**
- **`config/a-dot-lispworks.lisp`**

`config/configure.lisp` is preloaded into the image before it is shipped. It contains settings governing fundamental issues like where to find the LispWorks run time folder structure, and so on. You can override these settings in your saved image or in your initialization file. You should read through **`configure.lisp`** .

`config/siteinit.lisp` contains any forms that are appropriate to the whole site but which are to be loaded afresh each time the image is started. The sample **`siteinit.lisp`** file distributed with LispWorks contains only the form:

```
(load-all-patches)
```

On startup, the image loads **`siteinit.lisp`** and your initialization file, in that order. The command line options **`-siteinit`** and **`-init`** can be used to specify loading of different files or to suppress them altogether. See the example in **9.4 Saving and testing the configured image**, below, and **9.5 Initializing LispWorks** for further details.

`private-patches/load.lisp` is loaded by **`load-all-patches`**, and should contain forms to load any private (named) patches that Lisp Support might send you.

`config/a-dot-lispworks.lisp` is a sample personal initialization file. You might like to copy this somewhere convenient and edit it to create your own initialization file.

Both **`configure.lisp`** and **`a-dot-lispworks.lisp`** are preloaded into the image before it is shipped, so if you are happy with the settings in these files, you need not change them. See the example in **9.4 Saving and testing the configured image**, below, and **9.5 Initializing LispWorks** for further details.

9.4 Saving and testing the configured image

It is not usually necessary to save an image merely to preload patches and your configuration, because these load very quickly on modern machines.

However, if you want to save an image to reduce startup time for a complex configuration (such as large application code) or to save a non-windowing image, then proceed as described in this section.

9.4.1 Create a configuration file

Make a copy of **`config\configure.lisp`** called **`C:\temp\my-configuration.lisp`**. When you have made any desired changes in **`my-configuration.lisp`** you can save a new LispWorks image, as described in **9.4.2 Create and use a save-image script**.

9.4.2 Create and use a save-image script

1. Create a configuration and saving script **`C:\temp\save-config.lisp`**, containing:

   ```
   (in-package "CL-USER")
   (load-all-patches)
   (load "C:/temp/my-configuration.lisp")
   (save-image "my-lispworks")
   ```

2. Change directory to the LispWorks installation directory, for example:

   ```
   C:


   cd %PROGRAMFILES%\LispWorks
   ```

3. Start the supplied image using the configuration script as the build file. For example:

   ```
   C:\Program Files (x86)\LispWorks>lispworks-8-0-0-x86-win32.exe -build C:\temp\save-config.lisp
   ```

If the image will not run at this stage, it is probably not finding a valid key.

Saving the image takes some time.

You can now use the new **`my-lispworks.exe`** image from the Windows Explorer, or you may choose to add a shortcut. The supplied image is not required after the configuration process has been successfully completed.

Do not try to save a new image over an image that is currently running. Instead, save an image under a unique name, and then, if necessary, replace the new image with the old one after the call to **`save-image`** has returned.

9.4.3 What to do if no image is saved

If the LispWorks splash screen appears briefly but no image is saved, then there is some error while loading the build script. To see the error message, run the command with output redirected to a file, for example:

```
C:\Program Files (x86)\LispWorks>lispworks-8-0-0-x86-win32.exe -build C:\temp\save-config.lisp >
C:\temp\output.txt
```

Look in the file **`c:\temp\output.txt`**.

9.4.4 Testing the newly saved image

You should now test the new LispWorks image. To test a configured version of LispWorks, do the following:

1. Start up the new image.

   The window-based environment should now initialize—during initialization a window displaying a copyright notice will appear on the screen.

   You may wish to work through some of the examples in the *LispWorks® User Guide and Reference Manual*, to further check that the configured image has been successfully built.

2. Test the load-on-demand system. In the Listener, type:

   ```
   CL-USER 1 > (inspect 1)
   ```

   Before information about the fixnum 1 is printed, the system should load the inspector from the **`load-on-demand`** directory.

You can quit the inspector by typing **`:q`**.

9.4.5 Saving a non-windowing image

For some purposes such as scripting it is convenient to have a LispWorks image that does not start the graphical programming environment.

To save an image which does not automatically start the GUI, use a script as described in **9.4.2 Create and use a save-image script** but pass the **`:environment`** argument to **`save-image`**. For example:

```
(save-image "my-tty-lispworks" :environment nil)
```

9.5 Initializing LispWorks

When LispWorks starts up, it looks for an initialization file to load. The name of the file is held in **`*init-file-name*`**, and is **`~/.lispworks`** by default. You can use **`cl:parse-namestring`** to see the expansion of this path. The file may contain any valid Lisp code.

You can load a different initialization file using the option **`-init`** in the command line, for example (all on one line):

```
C:\Program Files\LispWorks>lispworks-8-0-0-x86-win32.exe -init my-lisp-init
```

would make LispWorks load **`my-lisp-init.lisp`** as the initialization file instead of that named by **`*init-file-name*`**.

The loading of the siteinit file (located by default at **`config\siteinit.lisp`**) is similarly controlled by the **`-siteinit`** command line argument or
`*site-init-file-name*`.

You can start an image without loading any personal or site initialization file by passing a hyphen to the **`-init`** and **`-siteinit`** arguments instead of a filename:

```
C:\Program Files\LispWorks>lispworks-8-0-0-x86-win32.exe -init - -siteinit -
```

This starts the LispWorks image without loading any initialization file. It is often useful to start the image in this way when trying to repeat a suspected bug. You should always start the image without the default initialization files if you are intending to resave it.

In all cases, if the filename is present, and is not a hyphen, LispWorks tries to load it as a normal file by calling **`load`**. If the load fails, LispWorks prints an error report.

9.6 Loading CLIM 2.0

Load CLIM 2.0 into LispWorks 8.0 with:

```
(require "clim")
```

and the CLIM demos with:

```
(require "clim-demo")
```

rather than the clim loader files in the clim distribution (which were the entry points in LispWorks 3).

A configuration file to save an image with CLIM 2.0 preloaded would look something like this:

```
(in-package "CL-USER")
(load-all-patches)
(require "clim")
(save-image "C:\\path\\to\\clim-lispworks")
```

9.6.1 Running the CLIM demos

To run the demo software, enter the following in a listener:

```
(require "clim-demo")
(clim-demo:start-demo)
```

This displays a menu listing all the demos. Choose the demo you wish to see. More information about the demos is in section "The CLIM demos" of the *Common Lisp Interface Manager 2.0 User's Guide*.

9.7 The Common SQL interface

The Common SQL interface requires ODBC or one of the supported database types listed in section "Supported databases" of the *LispWorks® User Guide and Reference Manual*.

9.7.1 Loading the Common SQL interface

To load the Common SQL interface to use ODBC enter:

```
(require "odbc")
```

and at run time call:

```
(sql:initialize-database-type :database-type :odbc)
```

and then you can connect to any installed ODBC datasource.

To load the Common SQL interface to use MySQL, enter:

```
(require "mysql")
```

and at run time call:

```
(sql:initialize-database-type :database-type :mysql)
```

See the *LispWorks® User Guide and Reference Manual* for further information.

9.8 Common Prolog and KnowledgeWorks

Common Prolog is bundled with KnowledgeWorks rather than with LispWorks. KnowledgeWorks is loaded by using:

```
(require "kw")
```

See the *KnowledgeWorks and Prolog User Guide* for further instructions.

9.9 Runtime library requirement on Windows

LispWorks for Windows requires the Microsoft Visual Studio runtime library **`msvcr80.dll`**. The LispWorks installer installs this DLL if it is not present.

Applications you build with LispWorks for Windows also require this DLL, so you must ensure it is available on target machines.

10 Configuration on Linux, x86/x64 Solaris & FreeBSD

10.1 Introduction

This chapter explains how to get LispWorks up and running on Linux, x86/x64 Solaris or FreeBSD, having already installed it. If you have not done this, refer to **4 Installation on Linux**, **5 Installation on x86/x64 Solaris**, or **6 Installation on FreeBSD**.

It is more useful to have an image customized to suit your particular environment and work needs. You can do this—setting useful pathnames, loading libraries, and so on—and then save the image to create another that will be configured as you require whenever you start it up.

This chapter covers the following topics:

- **10.2 License keys**
- **10.3 Configuring your LispWorks installation**
- **10.4 Saving and testing the configured image**
- **10.5 Initializing LispWorks**
- **10.6 Loading CLIM 2.0**
- **10.7 The Common SQL interface**
- **10.8 Common Prolog and KnowledgeWorks**

10.2 License keys

LispWorks is protected against unauthorized copying and use by a simple key protection mechanism. LispWorks will not start up until it finds a file containing a valid key.

The image looks for a file **`lwlicense`** in the following places, in order:

- In the current working directory.
- In the directory containing the LispWorks executable.
- In the **`lib/8-0-0-0/config`** subdirectory of the LispWorks installation directory.

When the file **`lwlicense`** is found, it must contain a valid key for the current machine. If you try to run LispWorks without a valid key, a message will be printed reporting that no valid key was found, and LispWorks will exit.

10.3 Configuring your LispWorks installation

Once you have successfully installed and run LispWorks, you can configure it to suit your local conditions and needs, producing an image that is set up the way you want it to be every time you start it up.

10.3.1 Levels of configuration

There are two levels of configuration: configuring and resaving the image, thereby creating a new image that is exactly as you want it at startup, and configuring certain aspects of LispWorks as it starts up.

These two levels are available for good reason: while some configuration details may be of use to all LispWorks users on your site (for instance, having a particular library built in to the image where before it was only load-on-demand) others may be a matter of personal preference (for instance how many editor windows are allowed on-screen, or the colors of tool windows).

In the first case, you use edited copies of files in the **`config`** directory to achieve your aims.

In the second case, you make entries in your initialization file. This is a file read every time LispWorks starts up, and it can contain any valid Common Lisp code. (Most of the configurable settings in LispWorks can be controlled from Common Lisp.) By default the file is called **`.lispworks`** and is in your home directory. Your initialization file can be changed via **`Tools > Preferences...`** in the LispWorks IDE.

10.3.2 Configuration files available

There are four sample configuration files in LispWorks library containing settings you can change in order to configure images:

- **`config/configure.lisp`**
- **`config/siteinit.lisp`**
- **`private-patches/load.lisp`**
- **`config/a-dot-lispworks.lisp`**

`config/configure.lisp` is preloaded into the image before it is shipped. It contains settings governing fundamental issues like where to find the LispWorks run time folder structure, and so on. You can override these settings in your saved image or in your initialization file. You should read through **`configure.lisp`** .

`config/siteinit.lisp` contains any forms that are appropriate to the whole site but which are to be loaded afresh each time the image is started. The sample **`siteinit.lisp`** file distributed with LispWorks contains only the form:

```
(load-all-patches)
```

On startup, the image loads **`siteinit.lisp`** and your initialization file, in that order. The command line options **`-siteinit`** and **`-init`** can be used to specify loading of different files or to suppress them altogether. See the example in **10.4 Saving and testing the configured image**, below, and **10.5 Initializing LispWorks** for further details.

`private-patches/load.lisp` is loaded by **`load-all-patches`**, and should contain forms to load any private (named) patches that Lisp Support might send you.

`config/a-dot-lispworks.lisp` is a sample personal initialization file. You might like to copy this into a file **`~/.lispworks`** in your home directory and edit it to create your own initialization file.

Both **`configure.lisp`** and **`a-dot-lispworks.lisp`** are preloaded into the image before it is shipped, so if you are happy with the settings in these files, you need not change them. See the example in **10.4 Saving and testing the configured image**, below, and **10.5 Initializing LispWorks** for further details.

10.4 Saving and testing the configured image

It is not usually necessary to save an image merely to preload patches and your configuration, because these load very quickly on modern machines.

However, if you want to save an image to reduce startup time for a complex configuration (such as large application code) or to save a non-windowing image, then proceed as described in this section.

10.4.1 Create a configuration file

Make a copy of **`config/configure.lisp`** called **`/tmp/my-configuration.lisp`**. When you have made any desired changes in **`my-configuration.lisp`** you can save a new LispWorks image, as described in **10.4.2 Create and use a save-image script**.

10.4.2 Create and use a save-image script

1. Create a configuration and saving script **`/tmp/save-config.lisp`**, containing:

   ```
   (in-package "CL-USER")
   (load-all-patches)
   (load "/tmp/my-configuration.lisp")
   (save-image "my-lispworks")
   ```

2. Change directory to the LispWorks installation directory, for example:

   ```
   % cd /usr/local/lib/LispWorks
   ```

3. Start the supplied image using the configuration script as the build file. For example:

   ```
   % lispworks-8-0-0-x86-linux -build /tmp/save-config.lisp
   ```

If the image will not run at this stage, it is probably not finding a valid key.

Saving the image takes some time.

You can now use the new **`my-lispworks`** image by starting it just as you did the supplied image. The supplied image is not required after the configuration process has been successfully completed.

Do not try to save a new image over an image that is currently running. Instead, save an image under a unique name, and then, if necessary, replace the new image with the old one after the call to **`save-image`** has returned.

10.4.3 Testing the newly saved image

You should now test the new LispWorks image. To test a configured version of LispWorks, do the following:

1. Change directory to **`/tmp`**.
2. Verify that your **`DISPLAY`** environment variable is correctly set and that your machine has permission to connect to the display.
3. Start up the new image.

   The window-based environment should now initialize—during initialization a window displaying a copyright notice will appear on the screen.

   You may wish to work through some of the examples in the *LispWorks® User Guide and Reference Manual*, to further check that the configured image has been successfully built.

4. Test the **`load-on-demand`** system. In the Listener, type:

   ```
   CL-USER 1 > (inspect 1)
   ```

   Before information about the fixnum 1 is printed, the system should load the inspector from the **`load-on-demand`** directory.

   You can quit the inspector by typing **`:q`**.

10.4.4 Saving a non-windowing image

For some purposes such as scripting it is convenient to have a LispWorks image that does not start the graphical programming environment.

To save an image which does not automatically start the GUI, use a script as described in **10.4.2 Create and use a save-image script** but pass the **`:environment`** argument to **`save-image`**. For example:

```
(save-image "my-tty-lispworks" :environment nil)
```

10.5 Initializing LispWorks

When LispWorks starts up, it looks for an initialization file to load. The name of the file is held in **`*init-file-name*`**, and is **`~/.lispworks`** by default. ~ denotes your home directory. The file may contain any valid Lisp code.

You can load a different initialization file using the option **`-init`** in the command line, for example:

```
% lispworks-8-0-0-x86-linux -init my-lisp-init
```

would make LispWorks load **`my-lisp-init.lisp`** as the initialization file instead of that named by **`*init-file-name*`**.

The loading of the siteinit file (located by default at **`config/siteinit.lisp`**) is similarly controlled by the **`-siteinit`** command line argument or
`*site-init-file-name*`.

You can start an image without loading any personal or site initialization file by passing a hyphen to the **`-init`** and **`-siteinit`** arguments instead of a filename:

```
% lispworks-8-0-0-x86-linux -init - -siteinit -
```

This starts the LispWorks image without loading any initialization file. It is often useful to start the image in this way when trying to repeat a suspected bug. You should always start the image without the default initialization files if you are intending to resave it.

In all cases, if the filename is present, and is not a hyphen, LispWorks tries to load it as a normal file by calling **`load`**. If the load fails, LispWorks prints an error report.

10.6 Loading CLIM 2.0

Load CLIM 2.0 into LispWorks 8.0 with:

```
(require "clim")
```

and the CLIM demos with:

```
(require "clim-demo")
```

rather than the clim loader files in the clim distribution (which were the entry points in LispWorks 3).

A configuration file to save an image with CLIM 2.0 preloaded would look something like this:

```
(in-package "CL-USER")
(load-all-patches)
(require "clim")
(save-image "/path/to/clim-lispworks")
```

10.6.1 Running the CLIM demos

To run the demo software, enter the following in a listener:

```
(require "clim-demo")
(clim-demo:start-demo)
```

This displays a menu listing all the demos. Choose the demo you wish to see. More information about the demos is in section "The CLIM demos" of the *Common Lisp Interface Manager 2.0 User's Guide*.

10.7 The Common SQL interface

The Common SQL interface requires ODBC or one of the supported database types listed in section "Supported databases" of the *LispWorks® User Guide and Reference Manual*.

10.7.1 Loading the Common SQL interface

To load the Common SQL interface to use ODBC enter:

```
(require "odbc")
```

and at run time call:

```
(sql:initialize-database-type :database-type :odbc)
```

and then you can connect to any installed ODBC datasource.

To load the Common SQL interface to use MySQL, enter:

```
(require "mysql")
```

and at run time call:

```
(sql:initialize-database-type :database-type :mysql)
```

See the *LispWorks® User Guide and Reference Manual* for further information.

10.8 Common Prolog and KnowledgeWorks

Common Prolog is bundled with KnowledgeWorks rather than with LispWorks. KnowledgeWorks is loaded by using:

```
(require "kw")
```

See the *KnowledgeWorks and Prolog User Guide* for further instructions.

10.9 Documentation on x86/x64 Solaris and FreeBSD

Except where explicitly mentioned, information stated as specific to LispWorks for Linux also applies to LispWorks for x86/x64 Solaris and LispWorks for FreeBSD.

11 Troubleshooting, Patches and Reporting Bugs

This chapter discusses other issues that arise when installing and configuring LispWorks. It provides solutions for possible problems you may encounter, and it discusses the patch mechanism and the procedure for reporting bugs.

11.1 Troubleshooting

This section describes some of the most common problems that can occur on any platform during installation or configuration.

11.1.1 License key errors

LispWorks looks for a valid license key when it is started up. If a problem occurs at this point, LispWorks exits.

These are the possible problems:

- LispWorks cannot find or read the key.
- The key is incorrect.
- Your license has expired, making the key no longer valid.

On Linux, x86/x64 Solaris and FreeBSD, this is also a possible cause of the problem:

- The machine name has changed since LispWorks was installed.

On macOS, Linux, x86/x64 Solaris and FreeBSD, the key is expected to be stored in a keyfile, and an appropriate error message is printed at the terminal for each case. If this message does not help you to resolve the problem, report it to Lisp Support and include the terminal output.

On Windows, the key is expected to be stored in the Windows registry. If you cannot resolve the problem, export your HKEY_LOCAL_MACHINE\SOFTWARE\LispWorks registry tree and include this with your report to Lisp Support.

11.1.2 Failure of the load-on-demand system

Module files are in the modules directory **`lib/8-0-0-0/load-on-demand`** under **`*lispworks-directory*`**.

If loading files on demand fails to work correctly, check that the modules directory is present. If it is not, perhaps your LispWorks installation is corrupted.

Do not remove any files from the modules directory unless you are really certain they will never be required.

The supplied image contains a trigger which causes **`*lispworks-directory*`** to be set on startup and hence you should not need to change its value. Subsequently saved images do not have this trigger.

11.1.3 Build phase (delivery-time) errors

A common cause of errors seen while building (delivering) an application is running part of the application's run time initialization, or something else that assumes the application is already running.

One error sometimes seen is **`"Not yet multiprocessing."`** and other likely build phase errors include those arising from code that assumes something about the run time environment.

Such initializations should be done at the start of the run time phase, as described in "Separate run time initializations from the build phase" in the *Delivery User Guide*.

11.1.4 Memory requirements

To run the full LispWorks system, with its GUI, you will need around 30MB of swap space for the image and whatever else is necessary to accommodate your application.

We recommend that you routinely check the size of your image using **`cl:room`**, whether you see warning messages or not.

When running a large image, you may occasionally see:

```
<**> Failed to enlarge memory
```

printed to the standard output.

The message means that the LispWorks image is close to the limit: it attempted to expand one of the GC generations, but there was not enough swap space to accommodate the resulting growth in image size. When this happens, the garbage collector is invoked. It will usually manage to free the required space, but if it cannot then crashes may result. Therefore you should take action to reduce allocation or increase available memory when you see this message.

Check the size of the image, both by **`cl:room`** and by OS facilities (such as **`ps`** or **`top`** on *nix, Task Manager on Windows) to see if all the sizes are as expected. If there are large discrepancies, check them.

Occasionally, however, continued demand for additional memory will end up exhausting resources. You will then see the message above repeatedly, and there will be little or no other activity apparent in the image. At this point you should restart the image, or increase swap space. In cases where external libraries are mapped above LispWorks and inhibit its growth, you may be able to relocate LispWorks, as described under "Startup relocation" in the *LispWorks® User Guide and Reference Manual*.

11.1.5 Corrupted LispWorks executable

Programs which attempt to clean up your machine by automatically removing data they identify as unnecessary may accidentally corrupt your LispWorks executable, because they do not understand its format. This will prevent LispWorks from starting.

Examples are the **`prelink`** cron job on Linux and CleanMyMac on Macintosh. These particular programs should no longer affect LispWorks, but there may be similar utilities in use.

If corruption occurs check if it has been caused by a clean-up utility. If this is the case, firstly configure your clean-up utility to ignore LispWorks, and then reinstall LispWorks.

11.2 Troubleshooting on Windows

This section describes some of the most common problems that can occur during installation or configuration of LispWorks for Windows.

11.2.1 Private patches not loaded on Windows 7, 8 & 10

Modify **`private-patches\load.lisp`** only via the menu command **Help > Install Private Patches...** to avoid problems with redirected files.

If your LispWorks installation is in the **`%ProgramFiles%`** folder and you edit **`private-patches\load.lisp`** directly, then Windows starts to use a redirected private copy of **`load.lisp`**. **Help > Install Private Patches...** will not update this copy, and thus your new patches will not be loaded.

If this occurs, the solution is to delete the redirected copy of **`load.lisp`** from your user profile space. On Windows 8 the location is like this:

```
C:\Users\lw\AppData\Local\VirtualStore\Program Files (x86)\LispWorks\lib\8-0-0-0\private-patches\
```

11.3 Troubleshooting on macOS

This section describes some of the most common problems that can occur during installation or configuration of LispWorks for Macintosh.

If you're using the LispWorks image with the X11/Motif GUI, see also **11.7 Troubleshooting on X11/Motif** below for issues specific to X11/Motif.

11.3.1 Uninstall requires administrator on macOS

You must be logged on an as administrator in order to run **`uninstall.command`** to uninstall LispWorks. This is because it uses the **`sudo`** command.

11.4 Troubleshooting on Linux

This section describes some of the most common problems that can occur during installation or configuration of LispWorks for Linux.

See also **11.7 Troubleshooting on X11/Motif** below for issues specific to X11/Motif.

11.4.1 Processes hanging

Some versions of Linux have a broken pthreads library. To workaround this set the environment variable LD_ASSUME_KERNEL=2.4.19 before running LispWorks. LD_ASSUME_KERNEL allows using older versions of pthreads, some of which do not work.

LispWorks 8.0 supports any Linux distribution with glibc 2.6 or later.

11.4.2 RPM_INSTALL_PREFIX not set

On Linux, during installation of CLIM, Common SQL, LispWorks ORB or KnowledgeWorks from a secondary rpm file you may see a message similar to this:

```
# rpm --install tmp/lispworks-clim-8.0-1.i386.rpm
Environment variable RPM_INSTALL_PREFIX not set, setting it to /usr
LispWorks installation not found in /usr.
error: %pre(lispworks-clim-8.0-1) scriptlet failed, exit status 1
error:   install: %pre scriptlet failed (2), skipping lispworks-clim-8.0-1
#
```

This is only a problem when LispWorks itself was installed in a non-default location (that is, using the **--prefix** RPM option). You would then want to supply that same **--prefix** value when installing the secondary rpm. A bug in RPM means that a required environment variable **RPM_INSTALL_PREFIX** is not set automatically to the supplied value. We have seen this bug in RPM version 4.2, as distributed with Red Hat 8 and 9.

The workaround is to set this environment variable explicitly before installing the secondary rpm. For example, if LispWorks was installed like this:

```
rpm --install --prefix /usr/lisp lispworks-8.0-1.i386.rpm
```

then you would add CLIM like this (in C shell):

```
setenv RPM_INSTALL_PREFIX /usr/lisp
rpm --install --prefix /usr/lisp lispworks-clim-8.0-1.i386.rpm
```

11.4.3 Using multiple versions of Motif on Linux

The version of Open Motif required by LispWorks 8.0 with the Motif GUI may not be compatible with other applications (including LispWorks 4.2). It is however compatible with LispWorks 7.1, LispWorks 6.x, LispWorks 5.x, LispWorks 4.4 and 4.3, so you for example you should be able to run LispWorks 8.0 and LispWorks 7.1 simultaneously with either Open Motif installed.

While it is not supported for LispWorks 5.1 and later versions, you can still use Lesstif for LispWorks 5.0 and earlier - see the Installation Guide for that version for details.

You may wish to maintain multiple versions of the Motif/Lesstif libraries in order to run various applications simultaneously. However, because the filenames of the libraries can conflict, this can only be done by installing libraries in non-standard locations.

When a library has been installed in a non-standard location, you can set the environment variable **LD_LIBRARY_PATH** to allow an application to find that library. Specifically, if <*motiflibdir*> denotes the directory containing the Motif 2.2 or 2.3 file **libXm.so** then set **LD_LIBRARY_PATH** to include <*motiflibdir*>.

Note: to find out which version of libXm your LispWorks 8.0 image is actually using, look in the bug form. See **11.9.3 Generate a bug report template** for instructions on generating the bug form.

11.5 Troubleshooting on x86/x64 Solaris

This section describes some of the most common problems that can occur during installation or configuration of LispWorks for x86/x64 Solaris.

See also **12.17.1 Problems with CAPI on GTK+** and **11.7 Troubleshooting on X11/Motif**.

11.5.1 GTK+ version

GTK+ 2 (version 2.4 or higher) is required to run the LispWorks image as distributed.

11.6 Troubleshooting on FreeBSD

This section describes some of the most common problems that can occur during installation or configuration of LispWorks for FreeBSD.

See also **11.7 Troubleshooting on X11/Motif** below for issues specific to X11/Motif.

11.7 Troubleshooting on X11/Motif

This section describes some of the most common problems that can occur using the LispWorks X11/Motif GUI, which is available on Linux, FreeBSD and macOS.

11.7.1 Problems with the X server

Running under X11/Motif, LispWorks may print a message saying that it is unable to connect to the X server. Check that the server is running, and that the machine the image is running on is authorized to connect to it. (See the manual entry for command **`xhost(1)`**.)

On macOS, if you attempt to start the LispWorks X11/Motif GUI in Terminal.app, an error message **`Failed to open display NIL`** is printed. Instead, run LispWorks in X11.app.

11.7.2 Problems with fonts on Motif

LispWorks may print a message saying that it is unable to open a font and is using a default instead. The environment will still run but it may not always use the right font.

LispWorks comes configured with the fonts most commonly found with the target machine type. However the fonts supplied vary between implementations and installations. The fonts available on a particular server can be determined by using the **`xlsfonts(1)`** command. Fonts are chosen based on the X11 resources. See **11.7.6 X11/Motif resources** for more information.

It may be necessary to change the fonts used by LispWorks.

11.7.3 Problems with colors

Running under X11, on starting up the environment, or any tool within it, LispWorks may print a message saying that a particular color could not be allocated.

This problem can occur if your X color map is full. If this is the case, LispWorks cannot allocate all the colors that are specified in the X11 resources.

This may happen if you have many different colors on your screen, for instance when displaying a picture in the root window of your display.

Colors are chosen based on the X11 resources. See **11.7.6 X11/Motif resources** for more information.

To remove the problem, you can then change the resources (for example, by editing the file mentioned in **11.7.6 X11/Motif resources**) to reduce the number of colors LispWorks allocates.

11.7.4 Motif mnemonics and Alt

Mnemonic processing on Motif always uses **`mod1`**, so we disable mnemonics if that is Lisp's **`Meta`** modifier to allow the Emacs-style editor to work. (The accelerator code uses the same keyboard mapping check as the mnemonics so **`Alt`** accelerators would also get disabled if you had them.)

11.7.5 Non-standard X11/Motif key bindings

On X11/Motif, if you want Emacs-style keys **`Ctrl-n, Ctrl-p`** in LispWorks list panels such as the Editor's buffers view, add the following to the X11 resources (see **11.7.6 X11/Motif resources**):

```
!
! Enable Ctrl-n, Ctrl-p in list panels
Lispworks*XmList.translations: #override\n\
                Ctrl<Key>p : ListPrevItem()\n\
                Ctrl<Key>n : ListNextItem()
!
```

11.7.6 X11/Motif resources

When using X11/Motif, LispWorks reads X11 resources in the normal way, using the application class Lispworks. The file **`app-defaults/Lispworks`** is used to supply fallback resources. You can copy parts of this file to **`~/Lispworks`** or some other configuration-specific location if you wish to change these defaults, and similarly for **`app-defaults/GcMonitor`**.

11.7.7 Motif installation on macOS

When attempting to starting the LispWorks X11/Motif GUI when the required version of Motif is not installed, LispWorks prints the error message:

```
Error: Could not register handle for external module X-UTILITIES::CAPIX11:
dyld: /Applications/LispWorks 8.0/lispworks-8-0-0-macos64-universal-gtk can't open library: /usr/lo
cal/lib/libXm.4.dylib (No such file or directory, errno = 2)
.
```

Ensure you install Motif as described in **2.4.8.2 The X11 GTK+ and Motif GUIs**. Restart X11.app and LispWorks after installation of Motif.

11.8 Updating with patches

We sometimes issue patches for LispWorks by email or download.

11.8.1 Extracting simple patches

Save the email attachment to your disk.

See **11.8.3.2 Private patches** below about location of your private patches.

11.8.2 If you cannot receive email

If your site has neither email nor ftp access, and you want to receive patches, you should contact Lisp Support to discuss a suitable medium for their transmission.

11.8.3 Different types of patch

There are two types of patch sent out by Lisp Support, and they must be dealt with in different ways.

11.8.3.1 Public patches

Public patches are general patches made available to all LispWorks customers. These are typically released in bundles of multiple different patch files; each file has a number as its name. For example:

```
patches\system\0001\0001.ofasl (for x86 Windows)
patches/system/0001/0001.ufasl (for x86 Linux)
patches/system/0001/0001.sfasl (for x86 Solaris)
patches/system/0001/0001.ffasl (for x86 FreeBSD)
patches/system/0001/0001.rfasl (for 32-bit ARM Linux and Android)
patches\system\0001\0001.64ofasl (for x64 Windows)
patches/system/0001/0001.64ufasl (for amd64 Linux)
patches/system/0001/0001.64xfasl (for Intel Macintosh)
patches/system/0001/0001.64yfasl (for Apple silicon Macintosh and iOS Simulator)
patches/system/0001/0001.64sfasl (for amd64 Solaris)
patches/system/0001/0001.64ffasl (for amd64 FreeBSD)
patches/system/0001/0001.64rfasl (for 64-bit ARM Linux and iOS)
patches/system/0001/0001.64xcfasl (for 64-bit iOS Simulator)
```

On receipt of a new patch bundle your system manager should update each local installation according to the installation instructions supplied with the patch bundle. This will add files to the patches subdirectory and increment the version number displayed by LispWorks.

You should consider saving a new image with the latest patches pre-loaded, as described in **8.4 Saving and testing the configured image** (macOS), **9.4 Saving and testing the configured image** (Windows) or **10.4 Saving and testing the configured image** (Linux, x86/x64 Solaris or FreeBSD).

11.8.3.2 Private patches

LispWorks patches are generally released in cumulative bundles. Occasionally Lisp Support may send you individual patch binaries named for example **`my-patch`** to address a problem or implement a new feature in advance of bundled ('public') patch releases. Such patches have real names, rather than numbers, and must be loaded once they have been saved to disk. You will need to ensure that LispWorks will load your private patches on startup, after public patches have been loaded.

Private patch loading is controlled by the file:

```
lib/8-0-0-0/private-patches/load.lisp
```

`private-patches/` is the default location for private patches, and patch loading instructions sent to you will assume this location. Therefore, on receipt of a private patch **`my-patch.ufasl`**, the simplest approach is to place it here. For example, on macOS:

*<install>***`/LispWorks 8.0 (64-bit)/Library/lib/8-0-0-0/private-patches/my-patch.64xfasl`**

On Windows (but see note below about the **Install Private Patches...** command):

*<install>***`lib\8-0-0-0\private-patches\my-patch.ofasl`**

On Linux:

*<install>***`/lib/8-0-0-0/private-patches/my-patch.ufasl`**

You will receive a Lisp form needed to load such a patch, such as:

```
(LOAD-ONE-PRIVATE-PATCH "my-patch" :SYSTEM)
```

This form should be added to the **`flet`** form in the file:

```
    private-patches/load.lisp
```

immediately after the commented example there. **`load-all-patches`** loads this file, and hence all the private patches listed therein.

You may choose to save a reconfigured image with the new patch loaded - for details see the instructions in **8.4 Saving and testing the configured image** (macOS), **9.4 Saving and testing the configured image** (Windows), or **10.4 Saving and testing the configured image** (Linux, x86/x64 Solaris or FreeBSD). You can alternatively choose to load the patch file on startup. The option you choose will depend on how many people at your site will need access to the new patch, and how many will need access to an image without the patch loaded.

Note: On Windows, the correct way to install private patches is using the menu item **Help > Install Private Patches...**. Select the private patch file with the **Add** button and edit the **`private-patches/load.lisp`** in the lower pane to include the loading form supplied by Lisp Support immediately after the commented example there. Then click **Save Changes**, which will run a helper application that interacts with the Windows User Access Control mechanism to allow you to write the files into the protected Program Files folder.

11.9 Reporting bugs

If you discover a bug, in either the software or the documentation, you can submit a bug report by any of the following routes.

- email
- fax
- paper mail (post)
- telephone

The addresses are listed in **11.9.8 Send the bug report**. Please note that we much prefer email.

11.9.1 Check for existing fixes

Before reporting a bug, please ensure that you have the latest patches installed and loaded. Visit **www.lispworks.com/downloads/patch-selection.html** for the latest patch release.

If the bug persists, check the Lisp Knowledgebase at **www.lispworks.com/support/** for information about the problem - we may already have fixed it or found workarounds.

If you need informal advice or tips, try joining the LispWorks users' mailing list. Details are at **www.lispworks.com/support/lisp-hug.html**.

11.9.2 Performance Issues

If the problem is poor performance, you should use **`room`**, **`extended-time`** and **`profile`** to check what actually happens. See the *LispWorks® User Guide and Reference Manual* for details of these diagnostic functions and macros.

If this does not help you to resolve the problem, submit a report to Lisp Support (see next section) and attach the output of the diagnostics.

11.9.3 Generate a bug report template

Whatever method you want to use to contact us, choose **Help > Report Bug** from any tool, or use the command **`Meta+X Report Bug`**, or at a Lisp prompt, use **`:bug-form`**, for example:

```
:bug-form "foo is broken" :filename "bug-report-about-foo.txt"
```

All three methods produce a report template you can fill in. In the GUI environment we prefer you use the **Report Bug** command - do this from within the debugger if an error has been signalled.

The bug report template captures details of the Operating System and Lisp you are running, as well as a stack backtrace if your Lisp is in the debugger. There may be delays if you do not provide this essential information.

If the issue you are reporting does not signal an error, or for some other reason you are not able to supply a backtrace, we still want to see the bug report template generated from the relevant LispWorks image.

11.9.4 Add details to your bug report

Under 'Urgency' tell us how urgent the issue is for you. We classify reports as follows:

ASAP	A bug or missing feature that is stopping progress. Probably needs a private patch, possibly under a support contract, unless a workaround can be found.
Current Release	Either a fix in the next patch bundle or as a private patch, possibly under a support contract.
Next Release	A fix would be nice in the next minor release.
Future Release	An item for our wishlist.
None	Probably not a bug or feature request.

Tell us if the bug is repeatable. Add instructions on how to reproduce it to the 'Description' field of the bug report form.

Include any other information you think might be relevant. This might be your code which triggers the bug. In this case, please send us a self-contained piece of code which demonstrates the problem (this is much more useful than code fragments).

Include the output of the Lisp image. In general it is not useful to edit the output, so please send it as-is. Where output files are very large (> 2MB) and repetitive, the first and last 200 lines might be adequate.

If the problem depends on a source or resource file, please include that file with the bug report.

If the bug report falls into one of the categories below, please also include the results of a backtrace after carrying out the extra steps requested:

- If the problem seems to be compiler-related, set **`*compiler-break-on-error*`** to **`t`**, and try again.
- If the problem seems to be related to **`error`** or conditions or related functionality, trace **`error`** and **`conditions:coerce-to-condition`**, and try again.
- If the problem is in the LispWorks IDE, and you are receiving too many notifiers, set **`dbg:*full-windowing-debugging*`** to **`nil`** and try again. This will cause the console version of debugger to be used instead.
- If the problem occurs when compiling or loading a large system, call (**`toggle-source-debugging nil`**) and try again.
- If you ever receive any unexpected terminal output starting with the characters **`<**>`**, please send all of the output—however much there is of it.

> **Note:** terminal output is that written to `*terminal-io*`. Normally this is not visible when running the macOS native GUI or the Windows GUI, though it is displayed in a Terminal.app or MS-DOS window if necessary.

11.9.5 Reporting crashes

Very occasionally, there are circumstances where it is not possible to generate a bug report form from the running Lisp which has the bug. For example, a delivered image may lack the debugger, or the bug may cause lisp to crash completely. In such circumstances:

1. It is still useful for us to see a bug report form from your lisp image so that we can see your system details. Generate the form before your code is loaded or a broken call is made, and attach it to your report.
2. Create a file **`init.lisp`** which loads your code that leads to the crash.
3. Run LispWorks with **`init.lisp`** as the initialization file and with output redirected to a file. For example, on macOS:

   ```
   % "/Applications/LispWorks 8.0 (64-bit)/LispWorks (64-bit).app/Contents/MacOS/lispworks-8-0-0-
   macos64-universal" -init init.lisp > lw.out
   ```

   where **`%`** denotes a Unix shell prompt.

   On Windows:

   ```
   C:\> "Program Files\LispWorks\lispworks-8-0-0-x86-win32.exe" -init init.lisp > lw.out
   ```

   where **C:\>** denotes the prompt in a MS-DOS command window.

   On Linux:

   ```
   % /usr/bin/lispworks-8-0-0-x86-linux -init init.lisp > lw.out
   ```

   where **`%`** denotes a Unix shell prompt.
4. Attach the **`lw.out`** file to your report. In general it is not useful to edit the output of your Lisp image, so please send it as-is. Where output files are very large (> 2MB) and repetitive, the first and last 200 lines might be adequate.

11.9.6 Log Files

If your application writes a log file, add this to your report. If your application does not write a log file, consider adding it, since a log is always useful. The log should record what the program is doing, and include the output of **`(room)`** periodically, say every five minutes.

You can make the application write a bug form to a log file automatically by making your error handlers call **`dbg:log-bug-form`**.

11.9.7 Reporting bugs in delivered images

Some delivered executables lack the debugger. It is still useful for us to see a bug report template from your Lisp image that was used to build the delivered executable. If possible, load your code and call **`(require "delivery")`** then generate the template.

For bugs in delivered LispWorks images, the best approach is to start with a very simple call to **`deliver`**, at level 0 and with the minimum of delivery keywords (**`:interface :capi`** and **`:multiprocessing t`** at most). Then deliver at increasingly severe levels. Add delivery keywords to address specific problems you find (see the *Delivery User Guide*.for details. However, please note that you are not expected to need to add more than 6 or so delivery keywords: do contact us if you are adding more than this.)

11.9.8 Send the bug report

Email is usually the best way. Send your report to:

```
lisp-support@lispworks.com
```

When we receive a bug report, we will send an automated acknowledgment, and the bug will be entered into the LispWorks bug management system. The automated reply has a subject line containing for example:

```
(Lisp Support Call #12345)
```

Please be sure to include that cookie in the subject line of all subsequent messages concerning your report, to allow Lisp Support to track it.

If you cannot use email, please either:

- Fax to +44 870 2206189.
- Post to Lisp Support, LispWorks Ltd, St John's Innovation Centre, Cowley Road, Cambridge, CB4 0WS, England.
- Telephone: +44 1223 421860.

Note: It is *very important* that you include a *stack backtrace* in your bug report wherever applicable. See **11.9.3 Generate a bug report template** for details. You can always get a backtrace from within the debugger by entering `:bb` at the debugger prompt.

11.9.9 Sending large files

Note: Please check with Lisp Support in advance if you are intending to send very large (> 2MB) files via email.

11.9.10 Information for Personal Edition users

We appreciate feedback from users of LispWorks Personal Edition, and often we are able to provide advice or workarounds if you run into problems. However please bear in mind that this free product is unsupported. For informal advice and tips, try joining the LispWorks users mailing list. Details are at **www.lispworks.com/support/lisp-hug.html**.

11.10 Transferring LispWorks to a different machine

This section lists the steps necessary to transfer LispWorks license to another machine.

1. Install LispWorks on your new machine.
2. Add latest patch bundle.
3. If you received private patches (named patch files, in the `lib/8-0-0-0/private-patches` directory) for this version of LispWorks, move them and your `private-patches/load.lisp` file to the corresponding location in the new installation.
4. Test the new installation by running LispWorks and check the patch banner in the output of **Help > Report Bug**. It should be identical to the original installation. If it differs, check that the public patches have been installed and that you private patches have been moved to the new `private-patches` folder along with the `load.lisp` file.

Please note that the LispWorks EULA restricts multiple installations so you may need to remove the original installation. Check your license agreement if you are unsure: the text of the shrinkwrap agreement is in the file `lib/8-0-0-0/license.txt`.

Instructions for uninstalling LispWorks are in the per-platform chapters of this manual:

- **2.6 Uninstalling LispWorks for Macintosh**
- **3.3 Uninstalling LispWorks for Windows**
- **4.9 Uninstalling LispWorks for Linux**
- **5.8 Uninstalling LispWorks for x86/x64 Solaris**
- **6.9 Uninstalling LispWorks for FreeBSD**

Some operating systems provide ways to copy software to another machine. A copied LispWorks installation will not run. Please contact Lisp Support if you want to install your license to a copied installation of LispWorks.

12 Release Notes

12.1 Keeping your old LispWorks installation

You can install LispWorks 8.0 in the same directory as previous versions such as LispWorks 7.1. This is because most of the 8.0 files are stored in a subdirectory called **`lib/8-0-0-0`**.

Binaries produced by **`cl:compile-file`** in previous versions of LispWorks do not load into a LispWorks 8.0 image. You must recompile all your code with the LispWorks 8.0 compiler.

12.2 Updating your code for LispWorks 8.0

Check through these release notes for things you need to update in code that already works in LispWorks 7.1.

If you are updating code that works only in versions earlier than LispWorks 7.1, then you should also consult earlier release notes, which are available at **www.lispworks.com/documentation**.

12.2.1 Conditionalizing code for different versions of LispWorks

When conditionalizing code for different versions of LispWorks, make your code work in the latest version and then conditionalize with feature expressions if necessary, depending on which previous versions of LispWorks you want to support.

For example, use **`#-lispworks7`** rather than **`#+lispworks8`**. This makes it more likely that the code will work without changes when LispWorks 9 is released in future.

Use only documented features. For an example see "Conditionalization for LispWorks versions" in the entry for *features* in the *LispWorks® User Guide and Reference Manual*.

12.3 Platform support

12.3.1 LispWorks for Macintosh supports Apple silicon Macs natively

LispWorks for Macintosh now supports Apple silicon based Macs natively using the arm64 architecture. The supplied images are now universal binaries.

12.3.2 LispWorks for Macintosh is always 64-bit

LispWorks for Macintosh is only released as a 64-bit application now because Apple have dropped support for 32-bit applications since macOS, 10.15 Catalina.

12.3.3 Runtimes for Android

LispWorks for Android Runtime supports 64-bit ARM devices now (the **`arm64-v8a`** ABI), as well as x86 and x86_64 devices (designed mainly for the Android Emulator when running on a computer with an Intel CPU).

The example script run-lw-android.sh now builds 4 images: 32-bit and 64-bit for each of ARM and x86.

Support for Android SDK 23 "text relocations" has been added.

The function **`hcl:deliver-to-android-project`** now supports Android Studio 3 and no longer supports Eclipse. The **`:using-ndk`** keyword has been removed.

The classes in the **`com.lispworks`** Java package are now provided by the **`lispworks.aar`** file that distributed with LispWorks. In previous releases, they were provided by the file **`lispworks.jar`**.

12.3.4 Runtimes for iOS

LispWorks for iOS Runtime now only supports 64-bit devices, because that is what Apple supports.

You can now create iOS Simulator and iOS device runtimes from an Apple silicon Mac (and without needing to use QEMU).

The way that you include the Lisp runtime in an XCode project has changed slightly (see 17 iOS interface in the LispWorks® User Guide and Reference Manual for details).

12.3.5 FreeBSD 12.x support

LispWorks 8.0 supports FreeBSD 12.x and later and is supplied as a standard package file, in pkg(8) format. Older versions of FreeBSD are not supported.

12.3.6 SPARC Solaris and AIX no longer supported

LispWorks 8.0 is not supported on SPARC Solaris or AIX.

12.3.7 Running on 64-bit machines

As far as we know each of the 32-bit LispWorks implementations runs correctly in the 32-bit subsystem of the corresponding 64-bit platform.

12.3.8 Code signing LispWorks images

12.3.8.1 Signing of the distributed executable

On macOS, the LispWorks application bundle is signed in the name of LispWorks Ltd.

On Microsoft Windows, the LispWorks Personal Edition executable is signed in the name of LispWorks Ltd.

Other LispWorks editions are not signed, because of the complications around image saving and delivery that this would lead to.

12.3.8.2 Signing your development image

On Microsoft Windows and macOS you can sign a development image saved using **`hcl:save-image`** with the **`:split`** argument. On macOS, the **`:split`** argument should have value **`:resources`**.

12.3.8.3 Signing your runtime application

On Microsoft Windows and macOS you can sign a runtime executable or dynamic library which was saved using **`lispworks:deliver`** with the **`:split`** argument.

12.3.8.4 Required runtime entitlements on Apple silicon Macs

LispWorks for Macintosh requires certain runtime entitlements to run on Apple silicon Macs. See 13.3.7 Saving images and delivering on Apple silicon Macs in the LispWorks® User Guide and Reference Manual for details.

12.3.9 macOS universal binaries are supported again

The supplied LispWorks (64-bit) for Macintosh images are now universal binaries, which run the correct native architecture on arm64 (Apple silicon) and x86_64 (Intel) Macintosh computers by default.

A running Lisp image only supports one architecture, chosen when the image was started. On a x86_64 based Macintosh, this is always the x86_64 architecture. On an arm64 Macintosh, a running LispWorks image can be either the native arm64 architecture or the x86_64 architecture (using Rosetta 2).

Functions such as **`save-image`** and **`deliver`** create an image containing only the running architecture and functions that operate on fasl files such as **`compile-file`** and **`load`** only support the running architecture.

To build a universal binary application from LispWorks 8.0 for Macintosh, you will need to install LispWorks on an arm64 (Apple silicon) Macintosh computer.

The functions **`hcl:save-universal-from-script`**, **`hcl:create-universal-binary`**, **`hcl:building-main-architecture-p`** and **`hcl:building-universal-intermediate-p`** are either new or non longer deprecated and can be used to control building a universal binary.

12.3.10 macOS images are now split into two files by default

The supplied LispWorks (64-bit) for Macintosh images are now *split*, which means that the Lisp heap is split into a separate file, named by adding **`.lwheap`** to the name of the executable. In the appliction bundle, this is stored in the **`Resources`** directory.

In addition, the *split* argument to **`save-image`** and **`deliver`** now defaults to **`:default`**, which causes the new image to be split by default on macOS.

12.4 GTK+ window system

LispWorks uses GTK+ as the default window system for CAPI and the LispWorks IDE on Linux, FreeBSD and x86/x64 Solaris. GTK+ is also supported on macOS as an alternative to Cocoa. LispWorks requires GTK+ 2 (version 2.4 or higher).

A few known problems are documented on **12.17.1 Problems with CAPI on GTK+**.

12.4.1 Using Motif instead of GTK+

Use of Motif with LispWorks on Linux, FreeBSD, x86/x64 Solaris and macOS is deprecated, but it is available by:

```
(require "capi-motif")
```

To use LispWorks 8.0 with Motif you also need Imlib2 (on Linux, FreeBSD and macOS) or Imlib (on Solaris) installed, as described earlier in this manual.

12.4.2 X11/Motif requires Imlib2 except on Solaris

LispWorks 8.0 requires Imlib2 1.4.3 or later to use the Motif GUI on Linux, FreeBSD and macOS. Some older versions of LispWorks required Imlib, which is a different library and is still required on Solaris.

12.5 New CAPI features

See the *CAPI User Guide and Reference Manual* for more details of these, unless directed otherwise. This section is not relevant to LispWorks for Mobile Runtime.

12.5.1 New thread-safe function to force a redisplay part of an capi:output-pane

The new function **`capi:redisplay-element`** can be used to force part of an **`capi:output-pane`** to be redisplayed. It's first argument can be an **`capi:output-pane`** or a **`capi:pinboard-object`** and it is equivalent to calling **`gp:invalidate-rectangle`**, except that it can be called from any thread.

12.5.2 Row and column separators in list panels

The classes **`capi:list-panel`** and **`capi:multi-column-list-panel`** now support visible separators between rows or columns by the new **`:separators`** initarg, with value **`nil`** (the default), **`:both`**, **`t`**, **`:horizontal`** or **`:vertical`**.

12.5.3 Support for reorderable columns in capi:multi-column-list-panel on GTK

The class **`capi:multi-column-list-panel`** supports reorderable columns on GTK by the new initarg **`:reorderable-columns`** and the **`:reorderable`** keyword in the column specification. Reorderable columns can be reordered by dragging their header.

12.5.4 New :x-adjust initarg for capi:multi-column-list-panel

The class **`capi:multi-column-list-panel`** has a new initarg **`:x-adjust`**, which provides the default value of the **`:adjust`** keyword in the column specifications. Its value must be a list of the same length as the **`:columns`** initarg.

12.5.5 Specifying the initial selection in capi:prompt-with-list

To specify the initial selection in **`capi:prompt-with-list`**, you can supply the keyword arguments **`:selection`**, **`:selected-item`** or **`:selected-items`**. These keywords were present in previous releases but not documented.

12.5.6 Menus can now display with both images and text on Microsoft Windows

The class **`capi:menu`** now supports display of both images and text on Microsoft Windows, like it did in previous releases for GTK+ and Cocoa.

12.5.7 Support for dark themes in capi:interface

The class **`capi:interface`** has new initargs **`:color-mode`** and **`:color-mode-callback`** and accessors **`capi:top-level-interface-color-mode`** and **`capi:top-level-interface-color-mode-callback`** to support dark themes and application-defined changes based on the theme.

The new function **`capi:top-level-interface-dark-mode-p`** can be used to detect when an interface is using a dark themes.

12.5.8 Support for dark themes in capi:set-editor-parenthesis-colors

The function **`capi:set-editor-parenthesis-colors`** now has a keyword argument **`:dark-background-colors`**, which is a list of colors to use for parentheses when the background is dark.

12.5.9 Support for dark themes in capi:stacked-tree

The default colors used by the class **`capi:stacked-tree`** change when a dark theme is used. If specify *colors* or *color-function*, then you may need to take special action.

12.5.10 New capi:rich-text-pane callback on Windows called when the user clicks a link

The class **`capi:rich-text-pane`** has a new initarg **`:link-callback`**, which is a function to be called if the text contains a hyperlink and the user click on it. This is only implemented on Windows.

12.5.11 Adding additional filters in capi:list-panel and capi:filtering-layout

The class **`capi:list-panel`** has a new initarg **`:filter-added-filters`**, which adds additional filters that apply to the items of the panel.

The class **`capi:filtering-layout`** has a new initarg **`:added-filters`** that does the same thing.

The function **`capi:filtering-layout-match-object-and-exclude-p`** returns an extra value, which is a list of the added filters that have been selected by the user.

12.5.12 Coordinates for keyboard events in the input model take account of scrolling

The callbacks in the input-model of a **`capi:output-pane`** are called with the coordinates of the pointer. In previous releases, the coordinates that are passed to callbacks of characters and keys did not take into account scrolling on some platforms. On the other hand, callbacks associated with mouse events (button clicks and motion) always took scrolling into account. In LispWorks 8.0, the coordinates that are passed to callbacks of characters and keys always take into account of scrolling the same way as mouse event callbacks.

That means that, by default (when **`:pane-can-scroll`** is **`nil`** in an **`capi:output-pane`**), the coordinates that the callbacks get when the pointer is on some graphic element match the coordinates that were used to draw the element (assuming there is no graphics transform).

12.5.13 capi:current-pointer-position always takes account of scrolling in capi:output-pane

The coordinates that the function **`capi:current-pointer-position`** returns when called with a **`capi:output-pane`** now take account of scrolling on all platforms. In previous releases, some platforms did not take account of scrolling.

12.5.14 Forcing scroll bars to be visible on macOS

The class **`capi:output-pane`** has a new initarg **`:scroll-bar-type`** that allows you to force the scroll bars to be visible on macOS regardless of the setting in the System Preferences.

12.6 Other CAPI and Graphics Ports changes

This section is not relevant to LispWorks for Mobile Runtime.

12.6.1 Drawing to an output-pane outside the display-callback

Code that draws to an **`capi:output-pane`** should only be called from within the pane's **`:display-callback`**. On some platforms, notably macOS Big Sur and later, drawing from other contexts will not work.

12.7 More new features

For details of these, see the documentation in the *LispWorks® User Guide and Reference Manual*, unless a manual is referenced explicitly.

12.7.1 Package-local nicknames

LispWorks now supports package-local nicknames, with the same interface as other Common Lisp implementations. This includes the new functions **`hcl:add-package-local-nickname`**, **`hcl:package-local-nicknames`**, **`hcl:package-locally-nicknamed-by-list`**, **`hcl:remove-package-local-nickname`** and the defpackage option :local-nicknames.

12.7.2 Support for pinning objects while in foreign code

The new macro **`hcl:with-pinned-objects`** can be used to prevent certain types of object from being moved by the garbage collector while in foreign code.

The value of the *allocation* keyword argument to **`cl:make-array`** can be **`:pinnable`** to make an array that can be pinned using **`hcl:with-pinned-objects`**.

The function **`system:make-typed-aref-vector`** takes a new keyword argument *allocation* which gives you control of where the new vector is allocated.

12.7.3 Specialized complex number array representations

LispWorks now supports a specialized array representation for **`(complex single-float)`** and **`(complex double-float)`**.

12.7.4 Double-float complex number optimization in the compiler

The compiler now optimizes arithmetic for values of type **`(complex double-float)`**.

12.7.5 The console now supports external formats on non-Windows platforms

Characters read and written via the console (**`*terminal-io*`**) are now encoded in an external format that is determined by the operating environment. See 27.16 The console external format in the LispWorks® User Guide and Reference Manual.

The new function **`hcl:set-console-external-format`** can be used to override this.

12.7.6 Encoding file names on non-Windows platforms based on locale

LispWorks now checks the POSIX locale variables to determine the external format in which file names should be encoded. See 27.14.1 Encoding of file names and strings in OS interface functions in the LispWorks® User Guide and Reference Manual for details.

12.7.7 Operating system interfaces on non-Windows based on locale

The values in the function **`lispworks:environment-variable`** and the command line arguments and environment variables in the functions **`system:call-system`**, **`system:call-system-showing-output`**, **`system:open-pipe`** and **`system:run-shell-command`** are now encoded using the same external format as file names, as described in 27.14.1 Encoding of file names and strings in OS interface functions in the LispWorks® User Guide and Reference Manual.

The command line arguments of LispWorks (see 27.4 The Command Line in the LispWorks® User Guide and Reference Manual) are decoded using the same external format.

This change should not affect arguments and values that contain only ASCII characters.

12.7.8 system:open-pipe and system:run-shell-command work with external formats

The functions **`system:open-pipe`** and **`system:run-shell-command`** have a new keyword argument, *external-format*, which is the external format to use. On non-Windows platforms, when neither the *external-format* nor the *element-type* are supplied, the external format defaults to the format specified by the POSIX environemnt variables **`LC_ALL`**, **`LC_CTYPE`** or **`LANG`**. If you use **`system:open-pipe`** in previous versions of LispWorks without supplying *element-type* and you want it to continue to not process the data using an external format, then supply *element-type* with **`base-char`** if you want code to work on all versions of LispWorks.

12.7.9 Specifying a timeout for system:pipe-exit-status

The function **`system:pipe-exit-status`** has a new keyword argument, *timeout*, which gives the maximum time to wait for the exit status. This overrides the *wait* argument, which is deprecated now.

12.7.10 system:run-shell-command can now return a signal number

On non-Windows platforms, the function **`system:run-shell-command`** with non-nil value for *wait* now returns a second value indicating the signal number that terminated the command if any.

12.7.11 Support for the GB18030 character encoding

LispWorks now supports the GB18030 character encoding with by the **`:gb18030`** external-format. See 26.6 External Formats to translate Lisp characters from/to external encodings in the LispWorks® User Guide and Reference Manual.

12.7.12 Configurable named services for remote debugging

The ports used for remote debugging can now be controlled by registering named service. See 3.7.6 TCP port usage in remote debugging in the LispWorks® User Guide and Reference Manual for more details.

12.7.13 Error handling and callbacks when starting remote debugging

The functions **`dbg:start-ide-remote-debugging-server`** and **`dbg:start-client-remote-debugging-server`** now have keyword arguments *announce* and *error* like **`comm:start-up-server`**.

The function **`dbg:start-ide-remote-debugging-server`** also has a *connection-callback* which is called with arguments to indicate if the connection was successful.

12.7.14 Using SSL for remote debugging

The functions **`dbg:ide-connect-remote-debugging`**, **`dbg:start-ide-remote-debugging-server`**, **`dbg:configure-remote-debugging-spec`** and **`dbg:start-client-remote-debugging-server`** and the macro **`dbg:with-remote-debugging-spec`** now have a **`:ssl`** keyword that allows SSL to be used for remote debugging connections.

12.7.15 Using IPv6 for remote debugging

The functions **`dbg:ide-connect-remote-debugging`**, **`dbg:start-ide-remote-debugging-server`**, **`dbg:configure-remote-debugging-spec`** and **`dbg:start-client-remote-debugging-server`** and the macro **`dbg:with-remote-debugging-spec`** now have a **`:ipv6`** keyword that allows IPv6 to be used for remote debugging connections.

12.7.16 Identifying object allocation in the profiler

The new function **`hcl:profiler-tree-to-allocation-functions`** prints a tree of function calls where the roots are allocation functions, making it easier to see where allocation happens.

12.7.17 Ignoring time in the garbage collector during profiling

The *gc* argument to the function **`hcl:set-up-profiler`** has new value **`:exclude`** which causes the profiler to ignore samples that are taken GC operation is in progress.

12.7.18 Version checking in compile-file-if-needed

`hcl:compile-file-if-needed` now checks that the version of the fasl file matches the version of the image and recompile if it does not match.

12.7.19 OpenSSL version defaults to 1.1 on Windows

The default OpenSSL DLL names in LispWorks for Windows are now those from OpenSSL 1.1.

12.7.20 Support for SSL using Apple Security Framework

LispWorks now supports (and defaults to) using the Apple Security Framework to implement SSL on macOS and iOS.

To allow the choice of SSL implementation to be made at run time and to allow code to specify configuration options that work with either implementation, a new concept called SSL Abtract Contexts has been added. The new system class **`comm:ssl-abstract-context`** represents these contexts, which can be created by the new functions **`comm:create-ssl-server-context`** and **`comm:create-ssl-client-context`**. The new function **`reset-ssl-abstract-context`** can be used to clear any cached information in a **`comm:ssl-abstract-context`**.

The new accessor **`comm:ssl-default-implementation`** can be used to control which SSL implementation is used and the new function **`comm:ssl-implementation-available-p`** can be used to check if an implementation is available.

The function **`comm:ensure-ssl`** can been extended to take an **`:implementation`** keyword, which specifies the implementation to initialize.

The functions **`comm:open-tcp-stream`**, **`comm:attach-ssl`**, **`comm:create-async-io-state-and-connected-tcp-socket`**, **`comm:socket-stream`** and **`comm:accept-tcp-connections-creating-async-io-states`** have been extended to take a **`comm:ssl-abstract-context`** as the **`:ssl-ctx`** argument.

The new FLI type **`comm:ssl-context-ref`** represents Apple Security Framework contexts.

The function **`comm:set-verification-mode`** has been extended to take a **`comm:ssl-context-ref`** as the *ssl-ctx* argument.

12.7.21 Specifying and accessing SSL certificates

The new function **`comm:ssl-connection-read-certificates`** specifies certificates for a SSL conection (a **`comm:socket-stream`** or a **`comm:async-io-state`**) by reading them from a file.

The new function **`comm:ssl-connection-get-peer-certificates-data`** can be used to get data about the certificate from a SSL connection.

The new functions **`comm:ssl-connection-copy-peer-certificates`**, **`comm:release-certificates-vector`**, **`comm:release-certificate`**, **`comm:get-certificate-data`**, **`comm:get-certificate-common-name`** and **`comm:get-certificate-serial-number`** can be used by experts to access certificates directly. These certificates are foreign pointers of type **`comm:sec-certificate-ref`** in the Apple Security Framework and **`comm:x509-pointer`** in OpenSSL.

12.7.22 SSL certificate generalized time API

The new type **`comm:generalized-time`** and new functions **`comm:generalized-time-p`**, **`comm:make-generalized-time`**, **`comm:generalized-time-pprint`**, **`comm:generalized-time-string`** and **`comm:parse-printed-generalized-time`** can be used to manipulate generalized times, as used in SSL certificates.

12.7.23 Reading DH parameters from a file

The new function **`comm:ssl-connection-read-dh-params-file`** reads a DH parameters file.

12.7.24 Detecting the SSL protocol version

The new function **`comm:ssl-connection-protocol-version`** returns the SSL protocol version that is being used by a connection.

12.7.25 comm:open-tcp-stream now returns information about errors

When the function **`comm:open-tcp-stream`** returns **`nil`** due to an error making the TCP connection, it now also returns a second value, which is a **`condition`** that gives information about the error.

12.7.26 Listen on the same port with more than one socket

The functions **`comm:start-up-server`** and **`comm:accept-tcp-connections-creating-async-io-states`** have a new keyword *reuseport*, which allows you to listen on the same port by multiple sockets, by using the socket option **`SO_REUSEPORT`**.

12.7.27 New function to close a socket handle

The new function **`comm:close-socket-handle`** can be used to close a native socket handle.

12.7.28 Newly documented customization for socket I/O error signaling

The function **`comm:socket-error`** existed in previous versions of LispWorks but is now documented. You can implement methods specialized on your own subclasses **`comm:socket-stream`** to customize signalling for socket I/O errors.

12.7.29 New condition classes in the socket interface

Errors relating to certain problems when using sockets are now signaled with the new condition classes **`comm:socket-io-error`**, **`comm:socket-create-error`**, **`comm:ssl-verification-failure`** and **`comm:ssl-handshake-timeout`**.

12.7.30 New condition classes in the Java interface

The Java interface can now signal two new condition classes **`lw-ji:jobject-call-method-error`** and **`lw-ji:java-program-error`**.

12.7.31 Calling static or non-static methods in the Java interface

The new function **`lw-ji:jobject-call-method`** can be used to call a non-static Java method and the new function **`lw-ji:call-java-static-method`** can be used to call a static Java method. These functions are useful when a static and non-static method with the same name exist in a class, because **`lw-ji:call-java-method`** (which also calls the non-static method in this case) would be ambiguous.

The macro **`lw-ji:define-java-caller`** and the function **`lw-ji:setup-java-caller`** have a new keyword argument *static-p* that controls the same thing.

12.7.32 Making a non-virtual call to a method in the Java interface

The macro **`lw-ji:define-java-caller`** and the function **`lw-ji:setup-java-caller`** have a new keyword argument *non-virtual-p* that makes the call non-virtual. Note that this is not normal Java behaviour, and may lead to surprising effects.

The new function **`lw-ji:call-java-non-virtual-method`** can also be used for this.

12.7.33 lw-ji:define-java-caller and lw-ji:setup-java-caller can now return lw-ji:jobject

The macro **`lw-ji:define-java-caller`** and the function **`lw-ji:setup-java-caller`** have a new keyword argument *return-jobject* that controls whether to return a Lisp object or a **`lw-ji:jobject`** when the method signature return value type is **`java.lang.String`** or **`java.lang.Object`**.

12.7.34 Specifying a Java class loader for Lisp proxy objects

The functions **`lw-ji:make-lisp-proxy`** and **`lw-ji:make-lisp-proxy-with-overrides`** have a new keyword argument *class-loader* to override the **`ClassLoader`** to pass as the first argument to the Java method **`Proxy.newProxyInstance`** when making the Lisp proxy.

12.7.35 Access to JNI jvalue objects

The new FLI type descriptor **`lw-ji:jvalue`** corresponds to the JNI C type **`jvalue`**. The new functions **`lw-ji:jvalue-store-jboolean`**, **`lw-ji:jvalue-store-jbyte`**, **`lw-ji:jvalue-store-jchar`**, **`lw-ji:jvalue-store-jshort`**, **`lw-ji:jvalue-store-jint`**, **`lw-ji:jvalue-store-jlong`**, **`lw-ji:jvalue-store-jfloat`**, **`lw-ji:jvalue-store-jdouble`** and **`lw-ji:jvalue-store-jobject`** can be used to set values in a **`lw-ji:jvalue`**. In typical usage of the Java interface, you will not need to use **`lw-ji:jvalue`** at all.

12.7.36 Getting a backtrace from a Java throwable object

The new function **`lw-ji:get-throwable-backtrace-strings`** can be used to get the backtrace from a Java **`throwable`** object.

12.7.37 lw-ji:create-instance-jobject-list is now exported from lw-ji

The function **`lw-ji:create-instance-jobject-list`** that was documented in previous releases is now exported from the **`lw-ji`** package. It was missing due to a bug.

12.7.38 Controlling aspects of LispWorks initialization on Android

The new Java methods **`com.lispworks.Manager.setRuntimeLispHeapDir`**, **`com.lispworks.Manager.setLispTempDir`** and **`com.lispworks.Manager.setClassLoader`** can be used to control aspects of how LispWorks initializes on Android. You should consult LispWorks support if you believe you need to use these.

12.7.39 New error codes from the InitLispWorks C function

The C function **`InitLispWorks`** has two new error codes, -1408 and -1409.

12.7.40 Stricter meaning of the :link-transparency argument to cl:directory

In LispWorks 8.0 and newer, if the **`file-namestring`** of its *pathname* argument is a symbolic link pointing to a directory and its *link-transparency* argument is **`nil`**, then **`directory`** returns it as a file. In previous versions of LispWorks, it was returned as a directory. Calling **`file-directory-p`** on such a link still returns true, so if you need to check if it is a directory or not, then you need to check first. The simplest way is to check that **`file-namestring`** returns **`nil`**.

12.7.41 Checking whether a file is a symbolic link

The new function **`hcl:file-link-p`** can be used to check whether a file is a symbolic link.

12.7.42 Reading a file into an array of bytes

The new function **`hcl:file-binary-bytes`** can be used to create an array of bytes from the contents a file.

12.7.43 cl:read-sequence and cl:write-sequence now depend on cl:stream-element-type

The Common Lisp function **`cl:read-sequence`** and **`cl:write-sequence`** now use **`cl:stream-element-type`** to detect character and binary streams. Prior to LispWorks 8.0, there was specialized behaviour for **`fundamental-character-output-stream`** and **`fundamental-binary-output-stream`** that was used to choose between character and binary I/O.

12.7.44 Specializing cl:read-sequence and cl:write-sequence is now documented

The **`stream:stream-read-sequence`** and **`stream:stream-write-sequence`** are now documented as the methods called by **`cl:read-sequence`** and **`cl:write-sequence`** respectively. These methods existed prior to LispWorks 8.0 but were not documented and the supplied methods have changed as described in **12.7.43 cl:read-sequence and cl:write-sequence now depend on cl:stream-element-type**.

12.7.45 New functions to compare strings without checking the length

The new function **`hcl:string=-limited`** and **`hcl:string-equal-limited`** simplify comparison of strings where you do not know the length of them.

12.7.46 Newly documented macro if-let

The macro **`lispworks:if-let`** if like **`cl:if`** but also binds a variable to the value of the test form. It is newly documented in LispWorks 8.0, but has been available since LispWorks 6.0.

12.7.47 Scheduling a repeating timer relative to the current time

If one of the functions **`mp:schedule-timer`**, **`mp:schedule-timer-relative`**, **`mp:schedule-timer-milliseconds`** and **`mp:schedule-timer-relative-milliseconds`** is called with a timer that is not scheduled or has already expired and the *absolute-expiration-time* or *relative-expiration-time* argument is **`nil`** and the *repeat-time* argument is non-nil, then the timer is scheduled to the current time plus *repeat-time*. In previous versions, this would have signaled an error.

12.7.48 hcl:get-temp-directory no longer returns a truename

The function **`hcl:get-temp-directory`** now returns a pathname with the name and type components set to **`nil`**. In previous releases, it returned a truename, with these components set to **`:unspecific`**.

12.7.49 Source location for macros that group other definition

The macro **`dspec:define-form-parser`** can now be used to define a form parser for a macro that acts like an implicit **`progn`**. Such macros (for example, **`eval-when`**) are used in a source file to wrap other definitions in the file, but do not have a name themselves.

12.7.50 The precompiled-regexp system class

The system class **`lispworks:precompiled-regexp`** and its predicate **`lispworks:precompiled-regexp-p`** have been added.

Instances of **`lispworks:precompiled-regexp`** represent a precompiled regular expression. They are produced by the function **`lispworks:precompile-regexp`**, and are used by the functions **`lispworks:find-regexp-in-string`**, **`lispworks:regexp-find-symbols`**, **`lispworks:count-regexp-occurrences`** and **`editor:regular-expression-search`**.

12.7.51 "Lax whitespace" regexp searching

The functions **`lispworks:find-regexp-in-string`**, **`lispworks:count-regexp-occurrences`** and **`lispworks:precompile-regexp`** takes a new keyword argument *space-string*.

When *space-string* is non-nil, then a "Lax whitespace" search is performed. That means that any sequence of space characters in the pattern are effectively replaced by the regexp specified by *space-string*. See the documentation for

lispworks:find-regexp-in-string for more details.

12.7.52 New arguments to the parser function defined by defparser

The parser function defined by **parsergen:defparser** now takes two keyword arguments:

- *message-stream* specifies a stream for outputting messages that are produced during the parsing.
- *return-match-tree-p* allows the function to return a match tree describing the matches during the parsing.

See 21.3 Functions defined by defparser in the LispWorks® User Guide and Reference Manual for details.

12.7.53 New system class gesture-spec

The new system class **system:gesture-spec** has been added. The concept of gesture specs existed prior to LispWorks 8.0, but **system:gesture-spec** was an internal undocumented symbol.

12.7.54 Limiting the number of splits in split-sequence

The functions **lispworks:split-sequence**, **lispworks:split-sequence-if** and **lispworks:split-sequence-if-not** have a new **:count** keyword which limits the number of times that the sequence is split.

12.7.55 Writing messages to system log files

The new functions **hcl:write-to-system-log** and **hcl:format-to-system-log** can be used to write messages to the operating system log files.

12.8 IDE changes

This section describes new features and other changes in the LispWorks Integrated Development Environment (IDE).

See the *LispWorks IDE User Guide* for details of the features mentioned. This section is not relevant to LispWorks for Mobile Runtime.

12.8.1 Support for Dark mode on macOS

the LispWorks IDE now supports Dark mode in the default Cocoa interface on macOS.

12.8.2 Configurable external format for the Shell tool

On non-Windows platforms, the external format that is used for communicating with the shell in the Shell tool can be set in the preferences dialog. By default, the external format defaults to the format specified by the POSIX environemnt variables **LC_ALL**, **LC_CTYPE** or **LANG**.

Note that the above affect only Shell windows which are created after any change is made. Existing windows are not affected.

12.8.3 A Commands menu has been added

There is a now a **Commands** menu, both in the menu bar of editing interfaces (or the **Works** menu on Windows) and in the context menu of editing panes. You can use the **Commands** menu to invoke Editor commands of your choice, thus making commands that you frequently find useful available by mouse clicks. You can control which commands appear on the menu by choosing **Commands > Display Command List...**.

12.8.4 Showing IDE interfaces in the Windows Browser

There is now a checkbox on the **Components** tab of the preferences for the Window Browser that controls whether to show IDE interfaces or not. Deselecting it makes it easier to find your interfaces in the graph.

12.8.5 The Works menu when displaying user-defined interfaces on Windows

On non-Windows platforms, when you display one your own interfaces while running the LispWorks IDE, it gets an extra menu called **Works**, which allows you to perform development operations. This menu is not added in delivered applications.

On Windows, the addition of the menu was inconsistent. In LispWorks 8.0, the **Works** menu is added to your interfaces on Windows as well, except when the LispWorks IDE is set to **Separate windows sharing a menu bar** in the preferences or for interfaces inside an MDI window.

Note that you can stop the addition of the **Works** menu in the LispWorks IDE by passing **`:auto-menus nil`** when creating the interface.

12.8.6 Identifying object allocation in the Profiler tool

The Profiler tool has a new menu item **Show calls to allocation functions [inverted]** in the context menu of the **Call Tree** and **Stacked Tree** tabs to show an inverted tree where the allocation functions are the roots, making it easier to see where allocation happens.

12.8.7 The Profiler automatically displays the results after profiling

The Profiler tool now switches automatically to the **Stacked Tree** tab after profiling finishes. This can be controled by the new **When Code To Profile finishes profiling:** option in the Profiler tool's preferences.

12.8.8 New operations in the Cumulative tab of the Profiler

The context menu in the **Cumulative** tab of the Profiler now allows you to show the selected function as the root of a tree or show the calls to the function as an inverted tree. The tree is shown in the **Call Tree** or **Stacked Tree** tabs according to the set of **When setting a root in the Cumulative tab:** in the Preferences dialog.

12.8.9 Building universal binaries on macOS with the Application Builder

You can build a universal binary on an arm64 (Apple silicon) Macintosh computer, using the same script as was used to build a normal ("thin") image.

12.8.10 Customizing the string used for hidden comments in folded definitions

The string used for hidden comments in folded definitions can be customized in the **Editor Options** tab of the Editor's Preferences dialog. See 4.14 Definition folding in the Editor User Guide for an explanation of defintion folding.

The style of the replacement string for hidden comments can be changed via **Preferences... > Environment > Styles > Styles Colors And Attributes**.

12.8.11 Operating on previous results in the Listener

The results of expression evaluation in the Listener are output as marked objects (except for trivial objects). That means they have a special style, and you can operate on them by using the context menu and choosing items from the **Marked Object** submenu.

The style used to display marked objects is called **Marked Object** and can be changed via **Preferences... > Environment > Styles > Styles Colors And Attributes**.

12.9 Editor changes

This section describes new features and other changes in the LispWorks editor, which is used in the Editor tool of the LispWorks IDE.

See the *Editor User Guide* for details of these changes. This section is not relevant to LispWorks for Mobile Runtime.

12.9.1 Lax whitespace matches

Search commands in the Editor can now be configured to search for whitespace in a "lax" way like in GNU Emacs. This means that any sequence of spaces in the search string will match any consecutive whitespace in the text.

The new editor variables **`editor:isearch-lax-whitespace`**, **`editor:isearch-regexp-lax-whitespace`**, **`editor:replace-lax-whitespace`** and **`editor:replace-regexp-lax-whitespace`** control the default state of lax whitespace matching in various operations.

For incremental searches, you can toggle between lax and exact whitespace matching for the current operation by typing Meta-s #\Space.

The new editor variable **`editor:search-whitespace-regexp`** contains the regexp used to match whitespace in lax mode.

12.9.2 Unique buffer names based on the directory of the file

When you open more than one file with a given name but in different directories, the editor has to ensure that the buffers have unique names. In previous releases, the first buffer was named after the file and subsequent buffers were named after the file with an additional suffix **<***n*>, where *n* was 2, 3, 4 etc.

In LispWorks 8.0, the editor still uses the name of the file is that is unique and otherwise renames all buffer with conflicting names to be unique. By default, the unique buffer name has a suffix **<***dirs*> where *dirs* includes enough of the directory name to make it unique.

The new function **`editor:set-buffer-name-directory-delimiters`** can be used to control how the buffer name is adjusted for files that have the same name, or switch back to the numerical suffix used in previous releases.

12.9.3 Definition folding

The editor now supports "definition folding", which means making the body of a definition invisible, as well as the preceding lines up to the previous definition, Currently the implementation applies only to Lisp definitions. A line starting with an open bracket is regarded as the begining of a Lisp definition, and the matching closing bracket is its end. The folding only affects the way the text in the buffer is displayed on the screen, and have no effect on the buffer contents.

The new editor commands The new editor commands **Fold Buffer Definitions**, , **Unfold Buffer Definitions** and and **Toggle Current Definition Folding** change the definition folding of the current buffer.

12.9.4 Indentation of loop

Uses of the extended **loop** form are now indented by the Editor based on the clause structure. For example:

```
(loop for index below 10
      when (foo index)
        do (print index))
```

12.9.5 Control how files are loaded

The new function **editor:set-pathname-load-function** allows you to set a specific function that will be called when loading files with a given type using the Editor command allows you to set a specific function that will be called when loading files with a given type using the Editor command **Load File** and the **File > Load** menu item in the LispWorks IDE.

12.9.6 Reverting a buffer with a different external format

Sometimes the editor uses the wrong external format when you open a file. The new editor command Sometimes the editor uses the wrong external format when you open a file. The new editor command **Revert Buffer With External Format** allows you to reopen the file after selecting a specific external format.

12.9.7 Toggling between the main and Output tabs in a Listener or Editor

The editor command The editor command **Invoke Tool** can now be used to toggle between the main and **Output** tabs in a Listener or Editor by using the character for the tool itself (**l** or **e** respectively).

12.9.8 Editor Ctrl+[and Ctrl+] key bindings in Windows emulation mode

When **Editor key are like Microsoft Windows, menu bar via Alt key** is checked in the **Environment > Emulation** preferences, the keyboard shortcuts **Ctrl+[** and **Ctrl+]** now perform **Beginning of Defun** and **End of Defun** respectively.

12.10 Foreign Language interface changes

See the *Foreign Language Interface User Guide and Reference Manual* for details of these changes.

12.10.1 :allow-null now defaults to nil for foreign strings as documented

The function **fli:convert-from-foreign-string** now gives an error when the pointer is a null pointer and the *allow-null* argument is true. In previous releases, a null pointer was always converted to **nil**. Pass **:allow-null t** if you want the previous behavior.

The functions **fli:convert-to-foreign-string** and **fli:convert-to-dynamic-foreign-string** now give an error when the string is **nil** and the *allow-null* argument is false. In previous releases, **nil** converted to a foreign string with length 0. Pass a Lisp string with length 0 if you want the previous behavior.

12.10.2 Checking for a valid foreign type

The function **fli:valid-foreign-type-p** has been added as a predicate to check if its argument is a valid foreign type.

12.10.3 fli:incf-pointer and fli:decf-pointer signal an error for types of size 0

The functions **`fli:incf-pointer`** and **`fli:decf-pointer`** now signal an error if the type pointed to by the argument has size 0 (for example, a **`void`** type).

12.10.4 Support for the C99 _Bool type (stdbool.h)

The foreign type **`:boolean`** now supports the C99 _Bool by using the form **`(:boolean :standard)`**.

12.10.5 Control of when fli:install-embedded-module deletes it temporary file

The function **`fli:install-embedded-module`** now has a keyword argument *delay-delete* that controls the time of deletion of the temporary file that is created for the module. A new variable **`fli:*install-embedded-module-delay-delete*`** is used as the default value for the keyword.

12.10.6 Use of dlopen on macOS

LispWorks now uses **`dlopen`** to load foreign modules on macOS, like on other non-Windows platforms. The default value of the **`:dlopen-flags`** argument to the function **`fli:register-module`** is now **`t`** on all platforms.

12.11 Objective-C changes

This section applies only to Macintosh and iOS platforms. See the *LispWorks Objective-C and Cocoa Interface User Guide and Reference Manual* for details.

12.11.1 objc:can-invoke-p can now be used with the result of current-super

The function **`objc:can-invoke-p`** can now be called with the result of **`objc:current-super`** in an Objective-C method implementation to see if the superclass implements a method.

12.11.2 objc:objc-bool on Macs based on Apple silicon

On Macs based on Apple silicon, the Objective-C BOOL type is the C99 _Bool type, whereas on Intel-based Macs it is a signed byte. The effect of this change is limited, but you may need to add extra Foreign Language Interface templates to your application if you have any that use the FLI type **`objc:objc-bool`**. See 10.6.1 Foreign Language Interface templates in the Delivery User Guide for details of how to do this.

12.11.3 The :darwin-lw-objc foreign module has been removed

If you call **`objc:ensure-objc-initialized`** explicitly when initializing your application, then you should not include **`:darwin-lw-objc`** in the list supplied to the **`:modules`** argument in LispWorks 8.0. Previous releases needed this foreign module in applications that use the CAPI, but the module has been removed and written in Lisp.

12.12 Common SQL changes

12.12.1 New helper functions and macro for prepared statements

The new functions **`sql:prepared-statement-set-and-execute`**, **`sql:prepared-statement-set-and-execute*`**, **`sql:prepared-statement-set-and-query`** and **`sql:prepared-statement-set-and-query*`** can be used to set the variables of a **`sql:prepared-statement`** and then execute or query using it.

The new macro **`sql:with-prepared-statement`** execute codes with a variable bound to a new **`prepared-statement`** and destroys it afterwards.

12.12.2 Calling connect with :if-exists and without :name

An error is signaled now if the *name* argument to the function **`sql:connect`** is not supplied and the *if-exists* argument is supplied with any value except **`:new`**. This is because there is no way to find an existing connection unless *name* is supplied.

12.12.3 New condition class signaled by connect

The new condition class **`sql:sql-failed-to-connect-error`** can be signaled by **`sql:connect`** for a failure to connect to a SQL database server. It typically indicates an incorrect connection specification such as a bad user name.

12.12.4 Some missing LOB functions are now exported

The functions **`sql:ora-lob-get-chunk-size`** and **`sql:ora-lob-file-set-name`** are now exported. They have been documented since LispWorks 5.0, but the symbols were not exported.

They were missing due to a bug.

12.13 KnowledgeWorks changes

This section applies only when you have a license to run KnowledgeWorks. See the *KnowledgeWorks and Prolog User Guide* for details, unless a manual is referenced explicitly.

12.13.1 New phrase predicate

A **`phrase`** predicate has been added to Common Prolog, which is the standard Prolog way to call a rule defined with **`defgrammer`**.

12.14 Application delivery changes

See the *Delivery User Guide* for more details of the changes mentioned in this section.

12.14.1 New values for the :interrupt-function keyword

Additional values **`:quit`**, **`:ignore`** and **`:break`** have been added to the :interrupt-function **`lispworks:deliver`** keyword.

12.15 Other changes

12.15.1 Changes in *features*

`:lispworks8.0` amd **`:lispworks8`** are present, **`:lispworks7.1`** and **`:lispworks7`** are not.

For a full description including information about the features used to distinguish new LispWorks implementations and platforms, see the entry for **`cl:*features*`** in the *LispWorks® User Guide and Reference Manual*.

12.15.2 ASDF version

The supplied ASDF is now version 3.3.5.

Note that this version of ASDF no longer exports **`uiop:defun*`** and **`uiop:defgeneric*`**. If you are using an older version of the **`serapeum`** library (from Quicklisp or github) that uses **`uiop:defun*`** then will need to update your copy.

12.15.3 The loop macro no longer allows "finally do" or "finally return"

The **`loop`** macro no longer allows the **`return`** or **`do`** loop keywords to be part of the **`finally`** clause. Previous releases of LispWorks supported this as an undocumented extension to ANSI Common Lisp.

To be compliant with all versions of LispWorks, change:

```
(loop ... finally do (form))
(loop ... finally return (form))
(loop named foo ... finally return (form))
```

to:

```
(loop ... finally (form))
(loop ... finally (return (form)))
(loop named foo ... finally (return-from foo (form)))
```

12.15.4 The loop macro now allows "of-type" with any atomic type

For compatibility with other implementations of the **`loop`** macro, you can now use any atomic type specifier after the **`of-type`** clause. Previous releases only allowed the types **`fixnum`**, **`float`**, **`t`** and **`nil`** as defined by ANSI Common Lisp.

12.15.5 Compiler macros are no longer expanded by the setf macro

When compiling a form like:

```
(setf (foo) value)
```

any compiler macro for **`foo`** is now ignored. If the **`setf`** form expands to a call to **`(setf foo)`** then any compiler macro for **`(setf foo)`** will be used. Compiler macros are defined using **`define-compiler-macro`**. This change does not affect macros defined by **`defmacro`**.

In previous releases, the compiler macro for **`foo`** would be used and any compiler macro for **`(setf foo)`** would be ignored.

12.15.6 hcl:fast-directory-files for a non-wild pathname

When the function **`hcl:fast-directory-files`** is called with a pathname whose name and type are both not wild, it now calls the callback with the file that matches that name and type. In previous releases, it called the callback for all files in the same directory, which was a bug.

12.15.7 cl:type-of now returns more specific types

The function **`type-of`** now returns more specific types for integers, characters, functions, keywords and the symbol **`t`**.

12.15.8 Loading old data files

Binary files created with **`hcl:dump-forms-to-file`** or **`hcl:with-output-to-fasl-file`** in LispWorks 7.1, LispWorks 7.0, LispWorks 6.1, LispWorks 6.0, LispWorks 5.x, LispWorks 4.4 or LispWorks 4.3 can be loaded into LispWorks 8.0 using **`system:load-data-file`**.

12.16 Documentation changes

12.16.1 Hyperlinks between manuals

The HTML documentation now has hyperlinks between manuals.

12.16.2 The HTML documentation directory

The HTML documentation is now in a directory named **`html-m`**, **`html-w`** or **`html-u`** according to which LispWorks platform you have installed. In older versions of LispWorks, this directory was named **`online`**.

12.16.3 Regular expression syntax

The documentation for the LispWorks regular expression syntax has been moved to the *LispWorks® User Guide and Reference Manual*.

12.16.4 Physical pathnames in LispWorks

Some details of how physical pathnames are parsed and printed in LispWorks are now documented in 27.18 Physical pathnames in LispWorks in the LispWorks® User Guide and Reference Manual.

12.16.5 New self-contained examples

These examples are entirely new:

```
(example-edit-file "capi/applications/interface-color-mode.lisp")
(example-edit-file "capi/applications/simple-othello.lisp")
(example-edit-file "capi/choice/filter-added-filters.lisp")
(example-edit-file "capi/choice/list-panel-keyboard-search.lisp")
(example-edit-file "capi/choice/stacked-tree.lisp")
(example-edit-file "capi/choice/tree-view-with-state.lisp")
(example-edit-file "capi/output-panes/modifier-change.lisp")
(example-edit-file "compiler/float-optimization.lisp")
(example-edit-file "editor/advanced/isearch-open-invisible.lisp")
(example-edit-file "editor/advanced/overlay-strings.lisp")
(example-edit-file "ssl/openssl-certificates.lisp")
```

```
(example-edit-file "ssl/openssl-server.lisp")
```

The Android Othello demo has been converted to an Android Studio project. Use with Eclipse is no longer supported.

12.16.6 Removed self-contained examples

The **`capi/graphics/plot-directly`** and **`capi/output-panes/scroll-test.lisp`** examples have been removed because they relied on drawing outside the **`:display-callback`**, which is no longer supported.

The **`capi/graphics/ruler.lisp`** example has been removed because it did not document anything useful.

The example files **`editor/syntax-coloring/defsys.lisp`** and **`editor/syntax-coloring/pkg.lisp`** have been removed because **`editor/syntax-coloring/syntax-coloring.lisp`** is self-contained.

12.17 Known Problems

12.17.1 Problems with CAPI on GTK+

The **`capi:interface-override-cursor`** is ignored by **`capi:text-input-pane`** which always displays its usual I-beam cursor. This is due to a limitation in the way that text-input-pane is implemented by GTK.

The normal navigation gesture (**`Tab`**) is treated as an editor command in **`capi:editor-pane`** and IDE tools based on this. Instead, use **`Ctrl+Tab`** to navigate from an editor pane in GTK+.

In GTK+ versions older than 2.12, the value of **`capi:option-pane`** *enabled-positions* has no effect on the visible representation of the items. In later versions of GTK+, the disabled items are grayed out.

In GTK+ versions older than 2.12, **`capi:display-tooltip`** does not work. In version 2.12 and later, the **`:x`** and **`:y`** keyword arguments of **`capi:display-tooltip`** might not be handled.

12.17.2 Problems with LispWorks for Macintosh

The Motif GUI does not work "out of the box" with Fink because LispWorks does not look for **`libXm`** etc in **`/sw/lib/`**.

12.17.3 Problems with the LispWorks IDE on Cocoa

Multithreading in the CAPI is different from other platforms. In particular, all windows run in a single thread, whereas on other platforms there is a thread per window.

The debugger currently does not work for errors in Cocoa Event Loop or Editor Command Loop threads. However, there is a **Get Backtrace** button so you can obtain a backtrace and also a **Debug Snapshot** button which aborts from the error but displays a debugger with a copy (snapshot) of the stack where the error occurred.

The online documentation interface currently starts a new browser window each time.

Setting **`*enter-debugger-directly*`** to **`t`** can allow the undebuggable processes to enter the debugger, resulting in the UI freezing.

Inspecting a long list (for example, 1000 items) via the Listener's **Inspect Star** editor command prompts you about truncation in a random window. If you cancel, the Inspector is still displayed.

The **Definitions > Compile** and **Definitions > Evaluate** menu options cause multiple "Press space to continue" messages to be displayed and happen interleaved rather than sequentially.

The **Buffers > Compile** and **Buffers > Evaluate** menu options cause multiple "Press space to continue" messages to be displayed and happen interleaved rather than sequentially.

12.17.4 Problems with CAPI and Graphics Ports on Cocoa

The **`capi:interface-override-cursor`** is ignored.

Some graphics state parameters are ignored, in particular *operation*, *stipple*, *pattern* and *fill-style*.

LispWorks ignores the System Preferences setting for the smallest font size to smooth.

There is no support for state images or checkboxes in **`capi:tree-view`**.

`capi:with-page` does not work, because Cocoa tries to control page printing.

The **`:help-callback`** initarg is only implemented for the **`:tooltip`** value of the type argument.

The **`:visible-border`** initarg only works for scrolling panes.

Caret movement and selection setting in **`capi:text-input-pane`** is implemented, but note that it works only for the focussed pane.

`capi:docking-layout` does not support (un)docking.

There is no meta key in the input-model of **`capi:output-pane`**. Note that, in the editor when using Emacs emulation, the **`Escape`** key can be used as a prefix.

There has been no testing with 256 color displays.

Some pinboard code uses **`:operation boole-xor`** which is not implemented.

The default menu bar is visible when the current window has no menu bar.

`capi:tree-view` is slow for a large number (thousands) of items.

The editor displays decomposed characters as separate glyphs.

The **`:gap`** option is not supported for the columns of **`capi:multi-column-list-panel`**.

`capi:display-dialog` ignores the specified **`:x`** and **`:y`** coordinates of the dialog (for drop-down sheets the coordinates are not relevant, and for dialogs which are separate windows Cocoa forces the window to be in the top-center of the screen).

12.18 Binary Incompatibility

If you have binaries (fasl files) which were compiled using LispWorks 7.1 or previous versions, please note that these are not compatible with this release. Please recompile all your code with LispWorks 8.0.

Index

A

accept-tcp-connections-creating-async-io-states function *12.7.20 : Support for SSL using Apple Security Framework* 79, *12.7.26 : Listen on the same port with more than one socket* 79

accessors

interface-override-cursor *12.17.1 : Problems with CAPI on GTK+* 91, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

top-level-interface-color-mode *12.5.7 : Support for dark themes in capi:interface* 74

top-level-interface-color-mode-callback *12.5.7 : Support for dark themes in capi:interface* 74

add-package-local-nickname function *12.7.1 : Package-local nicknames* 76

attach-ssl function *12.7.20 : Support for SSL using Apple Security Framework* 79

B

:boolean FLI type descriptor *12.10.4 : Support for the C99 _Bool type (stdbool.h)* 87

building-main-architecture-p function *12.3.9 : macOS universal binaries are supported again* 73

building-universal-intermediate-p function *12.3.9 : macOS universal binaries are supported again* 73

C

call-java-method function *12.7.31 : Calling static or non-static methods in the Java interface* 80

call-java-non-virtual-method function *12.7.32 : Making a non-virtual call to a method in the Java interface* 80

call-java-static-method function *12.7.31 : Calling static or non-static methods in the Java interface* 80

call-system function *12.7.7 : Operating system interfaces on non-Windows based on locale* 77

call-system-showing-output function *12.7.7 : Operating system interfaces on non-Windows based on locale* 77

can-invoke-p function *12.11.1 : objc:can-invoke-p can now be used with the result of current-super* 87

C functions

InitLispWorks *12.7.39 : New error codes from the InitLispWorks C function* 81

classes

docking-layout *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

editor-pane *12.17.1 : Problems with CAPI on GTK+* 91

filtering-layout *12.5.11 : Adding additional filters in capi:list-panel and capi:filtering-layout* 75

interface *12.5.7 : Support for dark themes in capi:interface* 74

list-panel *12.5.2 : Row and column separators in list panels* 74, *12.5.11 : Adding additional filters in capi:list-panel and capi:filtering-layout* 75

menu *12.5.6 : Menus can now display with both images and text on Microsoft Windows* 74

multi-column-list-panel *12.5.2 : Row and column separators in list panels* 74, *12.5.3 : Support for reorderable columns in capi:multi-column-list-panel on GTK* 74, *12.5.4 : New :x-adjust initarg for capi:multi-column-list-panel* 74, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

option-pane *12.17.1 : Problems with CAPI on GTK+* 91

output-pane *12.5.12 : Coordinates for keyboard events in the input model take account of scrolling* 75, *12.5.14 : Forcing scroll bars to* *be visible on macOS* 75, *12.6.1 : Drawing to an output-pane outside the display-callback* 76, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`rich-text-pane` *12.5.10 : New capi:rich-text-pane callback on Windows called when the user clicks a link* 75

`stacked-tree` *12.5.9 : Support for dark themes in capi:stacked-tree* 75

`text-input-pane` *12.17.1 : Problems with CAPI on GTK+* 91, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`tree-view` *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`close-socket-handle` function *12.7.27 : New function to close a socket handle* 80

`com.lispworks.Manager.setClassLoader` Java method *12.7.38 : Controlling aspects of LispWorks initialization on Android* 81

`com.lispworks.Manager.setLispTempDir` Java method *12.7.38 : Controlling aspects of LispWorks initialization on Android* 81

`com.lispworks.Manager.setRuntimeLispHeapDir` Java method *12.7.38 : Controlling aspects of LispWorks initialization on Android* 81

command line arguments

`-init` *11.9.5 : Reporting crashes* 68

`-init` on Linux *10.3.2 : Configuration files available* 54, *10.5 : Initializing LispWorks* 56

`-init` on macOS *8.3.3 : Configuration files available* 41, *8.5 : Initializing LispWorks* 44

`-init` on Windows *9.3.2 : Configuration files available* 48, *9.5 : Initializing LispWorks* 50

`-siteinit` on Linux *10.3.2 : Configuration files available* 54

`-siteinit` on macOS *8.3.3 : Configuration files available* 41

`-siteinit` on Windows *9.3.2 : Configuration files available* 48

`compile-file-if-needed` function *12.7.18 : Version checking in compile-file-if-needed* 78

condition classes

`java-program-error` *12.7.30 : New condition classes in the Java interface* 80

`jobject-call-method-error` *12.7.30 : New condition classes in the Java interface* 80

`socket-create-error` *12.7.29 : New condition classes in the socket interface* 80

`socket-io-error` *12.7.29 : New condition classes in the socket interface* 80

`sql-failed-to-connect-error` *12.12.3 : New condition class signaled by connect* 88

`ssl-handshake-timeout` *12.7.29 : New condition classes in the socket interface* 80

`ssl-verification-failure` *12.7.29 : New condition classes in the socket interface* 80

`configure-remote-debugging-spec` function *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

`connect` function *12.12.2 : Calling connect with :if-exists and without :name* 88, *12.12.3 : New condition class signaled by connect* 88

`convert-from-foreign-string` function *12.10.1 : :allow-null now defaults to nil for foreign strings as documented* 86

`convert-to-dynamic-foreign-string` function *12.10.1 : :allow-null now defaults to nil for foreign strings as documented* 86

`convert-to-foreign-string` function *12.10.1 : :allow-null now defaults to nil for foreign strings as documented* 86

corrupted executable *11.1.5 : Corrupted LispWorks executable* 60

`count-regexp-occurrences` function *12.7.51 : "Lax whitespace" regexp searching* 82

`create-async-io-state-and-connected-tcp-socket` function *12.7.20 : Support for SSL using Apple Security Framework* 79

`create-instance-jobject-list` function *12.7.37 : lw-ji:create-instance-jobject-list is now exported from lw-ji* 81

`create-ssl-client-context` function *12.7.20 : Support for SSL using Apple Security Framework* 78

`create-ssl-server-context` function *12.7.20 : Support for SSL using Apple Security Framework* 78

`create-universal-binary` function *12.3.9 : macOS universal binaries are supported again* 73

`current-pointer-position` function *12.5.13 : capi:current-pointer-position always takes account of scrolling in capi:output-pane* 75

`current-super` function *12.11.1 : objc:can-invoke-p can now be used with the result of current-super* 87

D

`:darwin-lw-objc` foreign module removed *12.11.3 : The :darwin-lw-objc foreign module has been removed* 87

`decf-pointer` function *12.10.3 : fli:incf-pointer and fli:decf-pointer signal an error for types of size 0* 87

`define-compiler-macro` macro *12.15.5 : Compiler macros are no longer expanded by the setf macro* 89

`define-form-parser` macro *12.7.49 : Source location for macros that group other definition* 82

`define-java-caller` macro *12.7.31 : Calling static or non-static methods in the Java interface* 80, *12.7.32 : Making a non-virtual call to a method in the Java interface* 80, *12.7.33 : lw-ji:define-java-caller and lw-ji:setup-java-caller can now return lw-ji:jobject* 80

`defparser` macro *12.7.52 : New arguments to the parser function defined by defparser* 83

`deliver` function *12.3.9 : macOS universal binaries are supported again* 73, *12.3.10 : macOS images are now split into two files by default* 73

`deliver-to-android-project` function *12.3.3 : Runtimes for Android* 72

`directory` function *12.7.40 : Stricter meaning of the :link-transparency argument to cl:directory* 81

`display-dialog` function *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`display-tooltip` function *12.17.1 : Problems with CAPI on GTK+* 91

`docking-layout` class *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`dump-forms-to-file` function *12.15.8 : Loading old data files* 90

E

editor commands

- Fold Buffer Definitions *12.9.3 : Definition folding* 85
- Invoke Tool *12.9.7 : Toggling between the main and Output tabs in a Listener or Editor* 86
- Load File *12.9.5 : Control how files are loaded* 86
- Revert Buffer With External Format *12.9.6 : Reverting a buffer with a different external format* 86
- Toggle Current Definition Folding *12.9.3 : Definition folding* 85
- Unfold Buffer Definitions *12.9.3 : Definition folding* 85

`editor-pane` class *12.17.1 : Problems with CAPI on GTK+* 91

editor variables

- **`isearch-lax-whitespace`** *12.9.1 : Lax whitespace matches* 85
- **`isearch-regexp-lax-whitespace`** *12.9.1 : Lax whitespace matches* 85
- **`replace-lax-whitespace`** *12.9.1 : Lax whitespace matches* 85
- **`replace-regexp-lax-whitespace`** *12.9.1 : Lax whitespace matches* 85
- **`search-whitespace-regexp`** *12.9.1 : Lax whitespace matches* 85

`ensure-objc-initialized` function *12.11.3 : The :darwin-lw-objc foreign module has been removed* 87

`ensure-ssl` function *12.7.20 : Support for SSL using Apple Security Framework* 78

`*enter-debugger-directly*` variable *12.17.3 : Problems with the LispWorks IDE on Cocoa* 91

environment-variable function *12.7.7 : Operating system interfaces on non-Windows based on locale* 77

errors while building application *11.1.3 : Build phase (delivery-time) errors* 60

errors while delivering application *11.1.3 : Build phase (delivery-time) errors* 60

extended-time macro *11.9.2 : Performance Issues* 66

F

Failed to enlarge memory *11.1.4 : Memory requirements* 60

fast-directory-files function *12.15.6 : hcl:fast-directory-files for a non-wild pathname* 90

features variable *12.2.1 : Conditionalizing code for different versions of LispWorks* 71, *12.15.1 : Changes in *features** 89

file-binary-bytes function *12.7.42 : Reading a file into an array of bytes* 81

file-link-p function *12.7.41 : Checking whether a file is a symbolic link* 81

fill-style graphics state parameter *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

filtering-layout class *12.5.11 : Adding additional filters in capi:list-panel and capi:filtering-layout* 75

filtering-layout-match-object-and-exclude-p function *12.5.11 : Adding additional filters in capi:list-panel and capi:filtering-layout* 75

finally clause in the **loop** macro no longer allows **do** or **return** *12.15.3 : The loop macro no longer allows "finally do" or "finally return"* 89

find-regexp-in-string function *12.7.51 : "Lax whitespace" regexp searching* 82

FLI type descriptors

- **:boolean** *12.10.4 : Support for the C99 _Bool type (stdbool.h)* 87
- **jvalue** *12.7.35 : Access to JNI jvalue objects* 81
- **objc-bool** *12.11.2 : objc:objc-bool on Macs based on Apple silicon* 87
- **sec-certificate-ref** *12.7.21 : Specifying and accessing SSL certificates* 79
- **ssl-context-ref** *12.7.20 : Support for SSL using Apple Security Framework* 79
- **x509-pointer** *12.7.21 : Specifying and accessing SSL certificates* 79

Fold Buffer Definitions editor command *12.9.3 : Definition folding* 85

format-to-system-log function *12.7.55 : Writing messages to system log files* 83

functions

- **accept-tcp-connections-creating-async-io-states** *12.7.20 : Support for SSL using Apple Security Framework* 79, *12.7.26 : Listen on the same port with more than one socket* 79
- **add-package-local-nickname** *12.7.1 : Package-local nicknames* 76
- **attach-ssl** *12.7.20 : Support for SSL using Apple Security Framework* 79
- **building-main-architecture-p** *12.3.9 : macOS universal binaries are supported again* 73
- **building-universal-intermediate-p** *12.3.9 : macOS universal binaries are supported again* 73
- **call-java-method** *12.7.31 : Calling static or non-static methods in the Java interface* 80
- **call-java-non-virtual-method** *12.7.32 : Making a non-virtual call to a method in the Java interface* 80
- **call-java-static-method** *12.7.31 : Calling static or non-static methods in the Java interface* 80
- **call-system** *12.7.7 : Operating system interfaces on non-Windows based on locale* 77
- **call-system-showing-output** *12.7.7 : Operating system interfaces on non-Windows based on locale* 77
- **can-invoke-p** *12.11.1 : objc:can-invoke-p can now be used with the result of current-super* 87
- **close-socket-handle** *12.7.27 : New function to close a socket handle* 80
- **compile-file-if-needed** *12.7.18 : Version checking in compile-file-if-needed* 78

configure-remote-debugging-spec *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

connect *12.12.2 : Calling connect with :if-exists and without :name* 88, *12.12.3 : New condition class signaled by connect* 88

convert-from-foreign-string *12.10.1 : :allow-null now defaults to nil for foreign strings as documented* 86

convert-to-dynamic-foreign-string *12.10.1 : :allow-null now defaults to nil for foreign strings as documented* 86

convert-to-foreign-string *12.10.1 : :allow-null now defaults to nil for foreign strings as documented* 86

count-regexp-occurrences *12.7.51 : "Lax whitespace" regexp searching* 82

create-async-io-state-and-connected-tcp-socket *12.7.20 : Support for SSL using Apple Security Framework* 79

create-instance-jobject-list *12.7.37 : lw-ji:create-instance-jobject-list is now exported from lw-ji* 81

create-ssl-client-context *12.7.20 : Support for SSL using Apple Security Framework* 78

create-ssl-server-context *12.7.20 : Support for SSL using Apple Security Framework* 78

create-universal-binary *12.3.9 : macOS universal binaries are supported again* 73

current-pointer-position *12.5.13 : capi:current-pointer-position always takes account of scrolling in capi:output-pane* 75

current-super *12.11.1 : objc:can-invoke-p can now be used with the result of current-super* 87

decf-pointer *12.10.3 : fli:incf-pointer and fli:decf-pointer signal an error for types of size 0* 87

deliver *12.3.9 : macOS universal binaries are supported again* 73, *12.3.10 : macOS images are now split into two files by default* 73

deliver-to-android-project *12.3.3 : Runtimes for Android* 72

directory *12.7.40 : Stricter meaning of the :link-transparency argument to cl:directory* 81

display-dialog *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

display-tooltip *12.17.1 : Problems with CAPI on GTK+* 91

dump-forms-to-file *12.15.8 : Loading old data files* 90

ensure-objc-initialized *12.11.3 : The :darwin-lw-objc foreign module has been removed* 87

ensure-ssl *12.7.20 : Support for SSL using Apple Security Framework* 78

environment-variable *12.7.7 : Operating system interfaces on non-Windows based on locale* 77

fast-directory-files *12.15.6 : hcl:fast-directory-files for a non-wild pathname* 90

file-binary-bytes *12.7.42 : Reading a file into an array of bytes* 81

file-link-p *12.7.41 : Checking whether a file is a symbolic link* 81

filtering-layout-match-object-and-exclude-p *12.5.11 : Adding additional filters in capi:list-panel and capi:filtering-layout* 75

find-regexp-in-string *12.7.51 : "Lax whitespace" regexp searching* 82

format-to-system-log *12.7.55 : Writing messages to system log files* 83

generalized-time-p *12.7.22 : SSL certificate generalized time API* 79

generalized-time-pprint *12.7.22 : SSL certificate generalized time API* 79

generalized-time-string *12.7.22 : SSL certificate generalized time API* 79

get-certificate-common-name *12.7.21 : Specifying and accessing SSL certificates* 79

get-certificate-data *12.7.21 : Specifying and accessing SSL certificates* 79

get-certificate-serial-number *12.7.21 : Specifying and accessing SSL certificates* 79

get-temp-directory *12.7.48 : hcl:get-temp-directory no longer returns a truename* 82

get-throwable-backtrace-strings *12.7.36 : Getting a backtrace from a Java throwable object* 81

ide-connect-remote-debugging *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

`incf-pointer` *12.10.3 : fli:incf-pointer and fli:decf-pointer signal an error for types of size 0* 87

`install-embedded-module` *12.10.5 : Control of when fli:install-embedded-module deletes it temporary file* 87

`invalidate-rectangle` *12.5.1 : New thread-safe function to force a redisplay part of an capi:output-pane* 74

`jobject-call-method` *12.7.31 : Calling static or non-static methods in the Java interface* 80

`jvalue-store-jboolean` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jbyte` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jchar` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jdouble` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jfloat` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jint` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jlong` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jobject` *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jshort` *12.7.35 : Access to JNI jvalue objects* 81

`load-data-file` *12.15.8 : Loading old data files* 90

`log-bug-form` *11.9.6 : Log Files* 68

`make-array` *12.7.2 : Support for pinning objects while in foreign code* 76

`make-generalized-time` *12.7.22 : SSL certificate generalized time API* 79

`make-lisp-proxy` *12.7.34 : Specifying a Java class loader for Lisp proxy objects* 80

`make-lisp-proxy-with-overrides` *12.7.34 : Specifying a Java class loader for Lisp proxy objects* 80

`make-typed-aref-vector` *12.7.2 : Support for pinning objects while in foreign code* 76

`open-pipe` *12.7.7 : Operating system interfaces on non-Windows based on locale* 77, *12.7.8 : system:open-pipe and system:run-shell-command work with external formats* 77

`open-tcp-stream` *12.7.20 : Support for SSL using Apple Security Framework* 79, *12.7.25 : comm:open-tcp-stream now returns information about errors* 79

`ora-lob-file-set-name` *12.12.4 : Some missing LOB functions are now exported* 88

`ora-lob-get-chunk-size` *12.12.4 : Some missing LOB functions are now exported* 88

`package-locally-nicknamed-by-list` *12.7.1 : Package-local nicknames* 76

`package-local-nicknames` *12.7.1 : Package-local nicknames* 76

`parse-printed-generalized-time` *12.7.22 : SSL certificate generalized time API* 79

`pipe-exit-status` *12.7.9 : Specifying a timeout for system:pipe-exit-status* 77

`precompiled-regexp-p` *12.7.50 : The precompiled-regexp system class* 82

`precompile-regexp` *12.7.51 : "Lax whitespace" regexp searching* 82

`prepared-statement-set-and-execute` *12.12.1 : New helper functions and macro for prepared statements* 88

`prepared-statement-set-and-execute*` *12.12.1 : New helper functions and macro for prepared statements* 88

`prepared-statement-set-and-query` *12.12.1 : New helper functions and macro for prepared statements* 88

`prepared-statement-set-and-query*` *12.12.1 : New helper functions and macro for prepared statements* 88

`profiler-tree-to-allocation-functions` *12.7.16 : Identifying object allocation in the profiler* 78

`prompt-with-list` *12.5.5 : Specifying the initial selection in capi:prompt-with-list* 74

`read-sequence` *12.7.43 : cl:read-sequence and cl:write-sequence now depend on cl:stream-element-type* 81, *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

`redisplay-element` *12.5.1 : New thread-safe function to force a redisplay part of an capi:output-pane* 74

register-module *12.10.6 : Use of dlopen on macOS* 87

release-certificate *12.7.21 : Specifying and accessing SSL certificates* 79

release-certificates-vector *12.7.21 : Specifying and accessing SSL certificates* 79

remove-package-local-nickname *12.7.1 : Package-local nicknames* 76

reset-ssl-abstract-context *12.7.20 : Support for SSL using Apple Security Framework* 78

room *11.9.2 : Performance Issues* 66

run-shell-command *12.7.7 : Operating system interfaces on non-Windows based on locale* 77, *12.7.8 : system:open-pipe and system:run-shell-command work with external formats* 77, *12.7.10 : system:run-shell-command can now return a signal number* 77

save-image *12.3.9 : macOS universal binaries are supported again* 73, *12.3.10 : macOS images are now split into two files by default* 73

save-universal-from-script *12.3.9 : macOS universal binaries are supported again* 73

schedule-timer *12.7.47 : Scheduling a repeating timer relative to the current time* 82

schedule-timer-milliseconds *12.7.47 : Scheduling a repeating timer relative to the current time* 82

schedule-timer-relative *12.7.47 : Scheduling a repeating timer relative to the current time* 82

schedule-timer-relative-milliseconds *12.7.47 : Scheduling a repeating timer relative to the current time* 82

set-buffer-name-directory-delimiters *12.9.2 : Unique buffer names based on the directory of the file* 85

set-console-external-format *12.7.5 : The console now supports external formats on non-Windows platforms* 76

set-editor-parenthesis-colors *12.5.8 : Support for dark themes in capi:set-editor-parenthesis-colors* 75

set-pathname-load-function *12.9.5 : Control how files are loaded* 86

setup-java-caller *12.7.31 : Calling static or non-static methods in the Java interface* 80, *12.7.32 : Making a non-virtual call to a method in the Java interface* 80, *12.7.33 : lw-ji:define-java-caller and lw-ji:setup-java-caller can now return lw-ji:jobject* 80

set-up-profiler *12.7.17 : Ignoring time in the garbage collector during profiling* 78

set-verification-mode *12.7.20 : Support for SSL using Apple Security Framework* 79

socket-error *12.7.28 : Newly documented customization for socket I/O error signaling* 80

socket-stream *12.7.20 : Support for SSL using Apple Security Framework* 79

split-sequence *12.7.54 : Limiting the number of splits in split-sequence* 83

split-sequence-if *12.7.54 : Limiting the number of splits in split-sequence* 83

split-sequence-if-not *12.7.54 : Limiting the number of splits in split-sequence* 83

ssl-connection-copy-peer-certificates *12.7.21 : Specifying and accessing SSL certificates* 79

ssl-connection-get-peer-certificates-data *12.7.21 : Specifying and accessing SSL certificates* 79

ssl-connection-protocol-version *12.7.24 : Detecting the SSL protocol version* 79

ssl-connection-read-certificates *12.7.21 : Specifying and accessing SSL certificates* 79

ssl-connection-read-dh-params-file *12.7.23 : Reading DH parameters from a file* 79

ssl-default-implementation *12.7.20 : Support for SSL using Apple Security Framework* 78

ssl-implementation-available-p *12.7.20 : Support for SSL using Apple Security Framework* 78

start-client-remote-debugging-server *12.7.13 : Error handling and callbacks when starting remote debugging* 77, *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

start-environment *2.5.3 : Start the Motif LispWorks GUI* 17

start-ide-remote-debugging-server *12.7.13 : Error handling and callbacks when starting remote debugging* 77, *12.7.13 : Error handling and callbacks when starting remote debugging* 78, *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

start-up-server *12.7.26 : Listen on the same port with more than one socket* 79

string=-limited *12.7.45 : New functions to compare strings without checking the length* 82

string-equal-limited *12.7.45 : New functions to compare strings without checking the length* 82

top-level-interface-dark-mode-p *12.5.7 : Support for dark themes in capi:interface* 74

type-of *12.15.7 : cl:type-of now returns more specific types* 90

valid-foreign-type-p *12.10.2 : Checking for a valid foreign type* 86

write-sequence *12.7.43 : cl:read-sequence and cl:write-sequence now depend on cl:stream-element-type* 81, *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

write-to-system-log *12.7.55 : Writing messages to system log files* 83

G

Garbage Collector message *11.1.4 : Memory requirements* 60

Garbage Collector output *11.1.4 : Memory requirements* 60

GC message *11.1.4 : Memory requirements* 60

GC output *11.1.4 : Memory requirements* 60

generalized-time type *12.7.22 : SSL certificate generalized time API* 79

generalized-time-p function *12.7.22 : SSL certificate generalized time API* 79

generalized-time-pprint function *12.7.22 : SSL certificate generalized time API* 79

generalized-time-string function *12.7.22 : SSL certificate generalized time API* 79

generic functions

stream-read-sequence *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

stream-write-sequence *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

gesture-spec system class *12.7.53 : New system class gesture-spec* 83

get-certificate-common-name function *12.7.21 : Specifying and accessing SSL certificates* 79

get-certificate-data function *12.7.21 : Specifying and accessing SSL certificates* 79

get-certificate-serial-number function *12.7.21 : Specifying and accessing SSL certificates* 79

get-temp-directory function *12.7.48 : hcl:get-temp-directory no longer returns a truename* 82

get-throwable-backtrace-strings function *12.7.36 : Getting a backtrace from a Java throwable object* 81

GTK *12.4 : GTK+ window system* 73

GTK+ *12.4 : GTK+ window system* 73

H

:help-callback initarg *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

I

IDE *12.8 : IDE changes* 83

ide-connect-remote-debugging function *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

if-let macro *12.7.46 : Newly documented macro if-let* 82

incf-pointer function *12.10.3 : fli:incf-pointer and fli:decf-pointer signal an error for types of size 0* 87

InitLispWorks C function *12.7.39 : New error codes from the InitLispWorks C function* 81

install-embedded-module function *12.10.5 : Control of when fli:install-embedded-module deletes it temporary file* 87

install-embedded-module-delay-delete variable *12.10.5 : Control of when fli:install-embedded-module deletes it temporary file* 87

Install Private Patches... menu command *11.2.1 : Private patches not loaded on Windows 7, 8 & 10* 61, *11.8.3.2 : Private patches* 66

Integrated Development Environment *12.8 : IDE changes* 83

`interface` class *12.5.7 : Support for dark themes in capi:interface* 74

`interface-override-cursor` accessor *12.17.1 : Problems with CAPI on GTK+* 91, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`invalidate-rectangle` function *12.5.1 : New thread-safe function to force a redisplay part of an capi:output-pane* 74

Invoke Tool editor command *12.9.7 : Toggling between the main and Output tabs in a Listener or Editor* 86

`isearch-lax-whitespace` editor variable *12.9.1 : Lax whitespace matches* 85

`isearch-regexp-lax-whitespace` editor variable *12.9.1 : Lax whitespace matches* 85

J

Java methods

  `com.lispworks.Manager.setClassLoader` *12.7.38 : Controlling aspects of LispWorks initialization on Android* 81

  `com.lispworks.Manager.setLispTempDir` *12.7.38 : Controlling aspects of LispWorks initialization on Android* 81

  `com.lispworks.Manager.setRuntimeLispHeapDir` *12.7.38 : Controlling aspects of LispWorks initialization on Android* 81

`java-program-error` condition class *12.7.30 : New condition classes in the Java interface* 80

`jobject-call-method` function *12.7.31 : Calling static or non-static methods in the Java interface* 80

`jobject-call-method-error` condition class *12.7.30 : New condition classes in the Java interface* 80

`jvalue` FLI type descriptor *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jboolean` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jbyte` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jchar` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jdouble` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jfloat` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jint` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jlong` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jobject` function *12.7.35 : Access to JNI jvalue objects* 81

`jvalue-store-jshort` function *12.7.35 : Access to JNI jvalue objects* 81

L

LispWorks fails to start *11.1.5 : Corrupted LispWorks executable* 60

LispWorks for Android Runtime *7.1 : Installing LispWorks for Android Runtime* 39

LispWorks for iOS Runtime *7.2 : Installing LispWorks for iOS Runtime* 39

LispWorks for Mobile Runtime *7 : Installation of LispWorks for Mobile Runtime* 39

LispWorks IDE tools

  Editor *12.9 : Editor changes* 85

`list-panel` class *12.5.2 : Row and column separators in list panels* 74, *12.5.11 : Adding additional filters in capi:list-panel and capi:filtering-layout* 75

`load-data-file` function *12.15.8 : Loading old data files* 90

Load File editor command *12.9.5 : Control how files are loaded* 86

`log-bug-form` function *11.9.6 : Log Files* 68

`loop` macro *12.15.3 : The loop macro no longer allows "finally do" or "finally return"* 89, *12.15.4 : The loop macro now allows "of-type" with any atomic type* 89

M

macros

- **`define-compiler-macro`** *12.15.5 : Compiler macros are no longer expanded by the setf macro* 89
- **`define-form-parser`** *12.7.49 : Source location for macros that group other definition* 82
- **`define-java-caller`** *12.7.31 : Calling static or non-static methods in the Java interface* 80, *12.7.32 : Making a non-virtual call to a method in the Java interface* 80, *12.7.33 : lw-ji:define-java-caller and lw-ji:setup-java-caller can now return lw-ji:jobject* 80
- **`defparser`** *12.7.52 : New arguments to the parser function defined by defparser* 83
- **`extended-time`** *11.9.2 : Performance Issues* 66
- **`if-let`** *12.7.46 : Newly documented macro if-let* 82
- **`loop`** *12.15.3 : The loop macro no longer allows "finally do" or "finally return"* 89, *12.15.4 : The loop macro now allows "of-type" with any atomic type* 89
- **`profile`** *11.9.2 : Performance Issues* 66
- **`with-output-to-fasl-file`** *12.15.8 : Loading old data files* 90
- **`with-page`** *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92
- **`with-pinned-objects`** *12.7.2 : Support for pinning objects while in foreign code* 76
- **`with-prepared-statement`** *12.12.1 : New helper functions and macro for prepared statements* 88
- **`with-remote-debugging-spec`** *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

`make-array` function *12.7.2 : Support for pinning objects while in foreign code* 76

`make-generalized-time` function *12.7.22 : SSL certificate generalized time API* 79

`make-lisp-proxy` function *12.7.34 : Specifying a Java class loader for Lisp proxy objects* 80

`make-lisp-proxy-with-overrides` function *12.7.34 : Specifying a Java class loader for Lisp proxy objects* 80

`make-typed-aref-vector` function *12.7.2 : Support for pinning objects while in foreign code* 76

`menu` class *12.5.6 : Menus can now display with both images and text on Microsoft Windows* 74

Motif *12.4 : GTK+ window system* 73

move LispWorks to another computer *11.10 : Transferring LispWorks to a different machine* 69

moving LispWorks to another computer *11.10 : Transferring LispWorks to a different machine* 69

`multi-column-list-panel` class *12.5.2 : Row and column separators in list panels* 74, *12.5.3 : Support for reorderable columns in capi:multi-column-list-panel on GTK* 74, *12.5.4 : New :x-adjust initarg for capi:multi-column-list-panel* 74, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

O

`objc-bool` FLI type descriptor *12.11.2 : objc:objc-bool on Macs based on Apple silicon* 87

`of-type` clause in the **`loop`** macro now allows any atomic type *12.15.4 : The loop macro now allows "of-type" with any atomic type* 89

`open-pipe` function *12.7.7 : Operating system interfaces on non-Windows based on locale* 77, *12.7.8 : system:open-pipe and system:run-shell-command work with external formats* 77

`open-tcp-stream` function *12.7.20 : Support for SSL using Apple Security Framework* 79, *12.7.25 : comm:open-tcp-stream now returns information about errors* 79

operation graphics state parameter *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`option-pane` class *12.17.1 : Problems with CAPI on GTK+* 91

`ora-lob-file-set-name` function *12.12.4 : Some missing LOB functions are now exported* 88

ora-lob-get-chunk-size function *12.12.4 : Some missing LOB functions are now exported* 88

output-pane class *12.5.12 : Coordinates for keyboard events in the input model take account of scrolling* 75, *12.5.14 : Forcing scroll bars to be visible on macOS* 75, *12.6.1 : Drawing to an output-pane outside the display-callback* 76, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

P

package-locally-nicknamed-by-list function *12.7.1 : Package-local nicknames* 76

package-local-nicknames function *12.7.1 : Package-local nicknames* 76

parse-printed-generalized-time function *12.7.22 : SSL certificate generalized time API* 79

pattern graphics state parameter *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

phrase predicate *12.13.1 : New phrase predicate* 88

pipe-exit-status function *12.7.9 : Specifying a timeout for system:pipe-exit-status* 77

poor performance *11.9.2 : Performance Issues* 66

precompiled-regexp system class *12.7.50 : The precompiled-regexp system class* 82

precompiled-regexp-p function *12.7.50 : The precompiled-regexp system class* 82

precompile-regexp function *12.7.51 : "Lax whitespace" regexp searching* 82

prepared-statement system class *12.12.1 : New helper functions and macro for prepared statements* 88

prepared-statement-set-and-execute function *12.12.1 : New helper functions and macro for prepared statements* 88

prepared-statement-set-and-execute* function *12.12.1 : New helper functions and macro for prepared statements* 88

prepared-statement-set-and-query function *12.12.1 : New helper functions and macro for prepared statements* 88

prepared-statement-set-and-query* function *12.12.1 : New helper functions and macro for prepared statements* 88

private patches

  not loaded on Windows *11.2.1 : Private patches not loaded on Windows 7, 8 & 10* 61

profile macro *11.9.2 : Performance Issues* 66

profiler-tree-to-allocation-functions function *12.7.16 : Identifying object allocation in the profiler* 78

prompt-with-list function *12.5.5 : Specifying the initial selection in capi:prompt-with-list* 74

R

read-sequence function *12.7.43 : cl:read-sequence and cl:write-sequence now depend on cl:stream-element-type* 81, *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

redisplay-element function *12.5.1 : New thread-safe function to force a redisplay part of an capi:output-pane* 74

Register... menu command *2.7 : Upgrading the LispWorks Edition* 17, *3.4 : Upgrading the LispWorks Edition* 20, *4.10 : Upgrading the LispWorks Edition* 28, *5.9 : Upgrading the LispWorks Edition* 33, *6.10 : Upgrading the LispWorks Edition* 38

register-module function *12.10.6 : Use of dlopen on macOS* 87

release-certificate function *12.7.21 : Specifying and accessing SSL certificates* 79

release-certificates-vector function *12.7.21 : Specifying and accessing SSL certificates* 79

remove-package-local-nickname function *12.7.1 : Package-local nicknames* 76

replace-lax-whitespace editor variable *12.9.1 : Lax whitespace matches* 85

replace-regexp-lax-whitespace editor variable *12.9.1 : Lax whitespace matches* 85

reset-ssl-abstract-context function *12.7.20 : Support for SSL using Apple Security Framework* 78

Revert Buffer With External Format editor command *12.9.6 : Reverting a buffer with a different external format* 86

rich-text-pane class *12.5.10 : New capi:rich-text-pane callback on Windows called when the user clicks a link* 75

room function *11.9.2 : Performance Issues* 66

run-shell-command function *12.7.7 : Operating system interfaces on non-Windows based on locale* 77, *12.7.8 : system:open-pipe and system:run-shell-command work with external formats* 77, *12.7.10 : system:run-shell-command can now return a signal number* 77

S

save-image function *12.3.9 : macOS universal binaries are supported again* 73, *12.3.10 : macOS images are now split into two files by default* 73

save-universal-from-script function *12.3.9 : macOS universal binaries are supported again* 73

schedule-timer function *12.7.47 : Scheduling a repeating timer relative to the current time* 82

schedule-timer-milliseconds function *12.7.47 : Scheduling a repeating timer relative to the current time* 82

schedule-timer-relative function *12.7.47 : Scheduling a repeating timer relative to the current time* 82

schedule-timer-relative-milliseconds function *12.7.47 : Scheduling a repeating timer relative to the current time* 82

search-whitespace-regexp editor variable *12.9.1 : Lax whitespace matches* 85

sec-certificate-ref FLI type descriptor *12.7.21 : Specifying and accessing SSL certificates* 79

set-buffer-name-directory-delimiters function *12.9.2 : Unique buffer names based on the directory of the file* 85

set-console-external-format function *12.7.5 : The console now supports external formats on non-Windows platforms* 76

set-editor-parenthesis-colors function *12.5.8 : Support for dark themes in capi:set-editor-parenthesis-colors* 75

set-pathname-load-function function *12.9.5 : Control how files are loaded* 86

setup-java-caller function *12.7.31 : Calling static or non-static methods in the Java interface* 80, *12.7.32 : Making a non-virtual call to a method in the Java interface* 80, *12.7.33 : lw-ji:define-java-caller and lw-ji:setup-java-caller can now return lw-ji:jobject* 80

set-up-profiler function *12.7.17 : Ignoring time in the garbage collector during profiling* 78

set-verification-mode function *12.7.20 : Support for SSL using Apple Security Framework* 79

socket-create-error condition class *12.7.29 : New condition classes in the socket interface* 80

socket-error function *12.7.28 : Newly documented customization for socket I/O error signaling* 80

socket-io-error condition class *12.7.29 : New condition classes in the socket interface* 80

socket-stream function *12.7.20 : Support for SSL using Apple Security Framework* 79

split-sequence function *12.7.54 : Limiting the number of splits in split-sequence* 83

split-sequence-if function *12.7.54 : Limiting the number of splits in split-sequence* 83

split-sequence-if-not function *12.7.54 : Limiting the number of splits in split-sequence* 83

sql-failed-to-connect-error condition class *12.12.3 : New condition class signaled by connect* 88

ssl-abstract-context system class *12.7.20 : Support for SSL using Apple Security Framework* 78

ssl-connection-copy-peer-certificates function *12.7.21 : Specifying and accessing SSL certificates* 79

ssl-connection-get-peer-certificates-data function *12.7.21 : Specifying and accessing SSL certificates* 79

ssl-connection-protocol-version function *12.7.24 : Detecting the SSL protocol version* 79

ssl-connection-read-certificates function *12.7.21 : Specifying and accessing SSL certificates* 79

ssl-connection-read-dh-params-file function *12.7.23 : Reading DH parameters from a file* 79

ssl-context-ref FLI type descriptor *12.7.20 : Support for SSL using Apple Security Framework* 79

ssl-default-implementation function *12.7.20 : Support for SSL using Apple Security Framework* 78

ssl-handshake-timeout condition class *12.7.29 : New condition classes in the socket interface* 80

ssl-implementation-available-p function *12.7.20 : Support for SSL using Apple Security Framework* 78

ssl-verification-failure condition class *12.7.29 : New condition classes in the socket interface* 80

stacked-tree class *12.5.9 : Support for dark themes in capi:stacked-tree* 75

start-client-remote-debugging-server function *12.7.13 : Error handling and callbacks when starting remote debugging* 77, *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

start-environment function *2.5.3 : Start the Motif LispWorks GUI* 17

start-ide-remote-debugging-server function *12.7.13 : Error handling and callbacks when starting remote debugging* 77, *12.7.13 : Error handling and callbacks when starting remote debugging* 78, *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

start-up-server function *12.7.26 : Listen on the same port with more than one socket* 79

stipple graphics state parameter *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

stream-read-sequence generic function *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

stream-write-sequence generic function *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

string=-limited function *12.7.45 : New functions to compare strings without checking the length* 82

string-equal-limited function *12.7.45 : New functions to compare strings without checking the length* 82

system classes

- **gesture-spec** *12.7.53 : New system class gesture-spec* 83
- **precompiled-regexp** *12.7.50 : The precompiled-regexp system class* 82
- **prepared-statement** *12.12.1 : New helper functions and macro for prepared statements* 88
- **ssl-abstract-context** *12.7.20 : Support for SSL using Apple Security Framework* 78

T

text-input-pane class *12.17.1 : Problems with CAPI on GTK+* 91, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

Toggle Current Definition Folding editor command *12.9.3 : Definition folding* 85

top-level-interface-color-mode accessor *12.5.7 : Support for dark themes in capi:interface* 74

top-level-interface-color-mode-callback accessor *12.5.7 : Support for dark themes in capi:interface* 74

top-level-interface-dark-mode-p function *12.5.7 : Support for dark themes in capi:interface* 74

transfer LispWorks to another computer *11.10 : Transferring LispWorks to a different machine* 69

transferring LispWorks to another computer *11.10 : Transferring LispWorks to a different machine* 69

tree-view class *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92, *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

type-of function *12.15.7 : cl:type-of now returns more specific types* 90

types

- **generalized-time** *12.7.22 : SSL certificate generalized time API* 79

U

uiop:defgeneric* removed from ASDF *12.15.2 : ASDF version* 89

uiop:defun* removed from ASDF *12.15.2 : ASDF version* 89

Unfold Buffer Definitions editor command *12.9.3 : Definition folding* 85

uninstalling LispWorks

- on FreeBSD *6.9 : Uninstalling LispWorks for FreeBSD* 38
- on Linux *4.9 : Uninstalling LispWorks for Linux* 28
- on Macintosh *2.6 : Uninstalling LispWorks for Macintosh* 17
- on Windows *3.3 : Uninstalling LispWorks for Windows* 20

on x86/x64 Solaris *5.8 : Uninstalling LispWorks for x86/x64 Solaris* 33

universal binaries

supported *12.3.9 : macOS universal binaries are supported again* 73

universal binary

supported *12.3.9 : macOS universal binaries are supported again* 73

V

`valid-foreign-type-p` function *12.10.2 : Checking for a valid foreign type* 86

variables

`*enter-debugger-directly*` *12.17.3 : Problems with the LispWorks IDE on Cocoa* 91

`*features*` *12.2.1 : Conditionalizing code for different versions of LispWorks* 71, *12.15.1 : Changes in *features** 89

`*install-embedded-module-delay-delete*` *12.10.5 : Control of when fli:install-embedded-module deletes it temporary file* 87

`:visible-border` initarg *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

W

window system *12.4 : GTK+ window system* 73

`with-output-to-fasl-file` macro *12.15.8 : Loading old data files* 90

`with-page` macro *12.17.4 : Problems with CAPI and Graphics Ports on Cocoa* 92

`with-pinned-objects` macro *12.7.2 : Support for pinning objects while in foreign code* 76

`with-prepared-statement` macro *12.12.1 : New helper functions and macro for prepared statements* 88

`with-remote-debugging-spec` macro *12.7.14 : Using SSL for remote debugging* 78, *12.7.15 : Using IPv6 for remote debugging* 78

`write-sequence` function *12.7.43 : cl:read-sequence and cl:write-sequence now depend on cl:stream-element-type* 81, *12.7.44 : Specializing cl:read-sequence and cl:write-sequence is now documented* 82

`write-to-system-log` function *12.7.55 : Writing messages to system log files* 83

X

`x509-pointer` FLI type descriptor *12.7.21 : Specifying and accessing SSL certificates* 79

Non-alaphanumerics

"Not yet multiprocessing." error *11.1.3 : Build phase (delivery-time) errors* 60