LispWorks® for the Windows® Operating System Common LispWorks® User Guide

Version 4.1



Copyright and Trademarks

Common LispWorks User Guide

Version 4.1

November 1998

Part number: 3LBDT2A15ND

Copyright © 1994–1998 by Harlequin Group plc.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Harlequin Group plc.

The information in this publication is provided for information only, is subject to change without notice, and should not be construed as a commitment by Harlequin Limited, Harlequin Incorporated, Harlequin Australia Pty. Limited, or Harlequin Group plc. Harlequin Group plc assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication. The software described in this book is furnished under license and may only be used or copied in accordance with the terms of that license.

LispWorks is a registered trademark of Harlequin Group plc. Delivery, Transducer/PC, The Authoring Book, ClassWorks, and Knowledge-Works are all trademarks of Harlequin Group plc.

Adobe and PostScript are registered trademarks of Adobe Systems Incorporated. Other brand or product names are the registered trademarks or trademarks of their respective holders.

The code for walker.lisp and compute-combination-points is excerpted with permission from PCL, Copyright © 1985, 1986, 1987, 1988 Xerox Corporation.

CLX and CLUE bear the following copyright notice, which applies to the parts of LispWorks derived therefrom:

Texas Instruments Incorporated, P.O. Box 149149, MS 2151, Austin, Texas 78714-9149

Copyright © 1987, 1988, 1989, 1990, 1991 Texas Instruments Incorporated.

Permission is granted to any individual or institution to use, modify and distribute this software, provided that this complete copyright and permission notice is maintained, intact, in all copies and documentation. Texas Instruments Incorporated provides this software "as is" without express or implied warranty.

The XP Pretty Printer bears the following copyright notice, which applies to the parts of LispWorks derived therefrom: Copyright © 1989 by the Massachusetts Institute of Technology, Cambridge, Massachusetts.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that this copyright and permission notice appear in all copies and supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representation about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. M.I.T. disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall M.I.T. be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

US Government Use

The LispWorks Software is a computer software program developed at private expense and is subject to the following Restricted Rights Legend: "Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in (i) FAR 52.227-14 Alt III or (ii) FAR 52.227-19, as applicable. Use by agencies of the Department of Defense (DOD) is subject to Harlequin's customary commercial license as contained in the accompanying license agreement, in accordance with DFAR 227.7202-1(a). For purposes of the FAR, the Software shall be deemed to be `unpublished' and licensed with disclosure prohibitions, rights reserved under the copyright laws of the United States. Harlequin Incorporated, One Cambridge Center, Cambridge, Massachusetts 02142."

Europe:	North America:	Electronic Access:
Harlequin Limited Barrington Hall Barrington Cambridge CB2 5RG	Harlequin Incorporated One Cambridge Center Cambridge, MA 02142 U.S.A.	http://www.harlequin.co.uk/ http://www.harlequin.com/
U.K. telephone +44 1223 873 800 fax +44 1223 873 873	telephone +1 617 374 2400 fax +1 617 252 6505	

Contents

Preface ix

1 Introduction 1

Major tools 2

2 A Short Tutorial 5

Starting the environment 5 Creating a listener 6 Using the debugger 8 Viewing output 11 Inspecting objects using the inspector 12 Examining classes in the class browser 15 Summary 17

3 Common Features 19

Loading and displaying tools 20 Setting global preferences 23 Quitting the environment 24 Performing editing functions 24 The history list 26 Operating on files 28 Displaying packages 29 Performing operations on selected objects 31 Using different views 34 Tracing symbols from tools 36 Linking tools together 37 Filtering information 38 Examining the window itself 39

4 Getting Help 41

Online manuals in HTML format 41 Online help for editor commands 44 Browsing manuals online using Adobe Acrobat 44

5 Manipulating Graphs 45

An overview of the graph view 45 Searching graphs 47 Expanding and collapsing graphs 47 Moving nodes in graphs 48 Displaying plans of graphs 48 Preferences for graph layouts 50

6 The LispWorks podium 57

7 The Class Browser 59

Simple use of the class browser 60 Examining slot information 67 Examining superclasses and subclasses 71 Examining classes graphically 74 Examining generic functions and methods 78 Examining initargs 81 Examining class precedences 83

8 The Compilation Conditions Browser 87

Introduction 87 Examining error conditions 88 Examining output 89 Configuring the display 90 Access to other tools 92

9 The Debugger Tool 93

Description of the debugger 95 What the debugger tool does 97 Simple use of the debugger tool 97 The stack in the debugger 98 An example debugging session 99 Performing operations on the error condition 101 Performing operations on frames in the stack 102 Performing operations on variables in a frame 102 Configuring the debugger tool 103

10 The Editor 107

Displaying and editing files 108 Displaying and swapping between buffers 112 Displaying Common Lisp definitions 114 Finding definitions 115 Changed definitions 115 Displaying output messages in the editor 115 Setting editor preferences 115 Getting started with the editor 117 Other essential commands 122 Cutting, copying and pasting using the clipboard 122 Cutting, copying and pasting using the kill ring 123 Searching and replacing text 127 Using Lisp-specific commands 129 Help with editing 134

11 The Function Call Browser 135

Introduction 135 Examining functions using the graph view 136 The Function Description button 137 Examining functions using the text view 139 Configuring the function call browser 141 Configuring graph displays 143 Performing operations on functions 144

12 The Generic Function Browser 145

Examining information about methods 145 Examining information about combined methods 149 Configuring the generic function browser 153

13 The Inspector 155

Examining objects 157 Filtering the display 160 Operating upon objects and items 162 Configuring the inspector 166 Customizing the inspector 168 Creating new inspection formats 169

14 Example: Using The Interface Builder 173

Creating the basic layout 175 Specifying attribute values 177 Creating the menu system 180 Specifying callbacks in the interface definition 183 Saving the interface 186 Defining the callbacks 186 Creating a system 189 Testing the example interface 189

15 The Interface Builder 191

Creating or loading interfaces 193 Creating an interface layout 196 Creating a menu system 199 Editing and saving code 204 Performing operations on objects 208 Performing operations on the current interface 213

16 The Listener 217

The basic features of a listener 217 Evaluating simple forms 219 Re-evaluating forms 220 Interrupting evaluation 221 The History menu 221 The Expression submenu 222 The Values submenu 223 The Debug menu 223 Execute mode 224 Help with editing in the listener 225

17 The Output Browser 227

18 The Process Browser 231

Displaying and sorting the output 232 Process control 233 Other ways of breaking processes 234 Updating the process browser 234

19 The Profiler 235

Introduction 235 A description of profiling 238 Steps involved in profiling code 238 Format of the results 243 Interpreting the results 244 Some examples 245

20 The System Browser 249

Introduction 249 Defining a system 250 A description of the system browser 252 Examining the system graph 253 Examining systems in the text view 256 Generating and executing plans in the preview view 258 Examining output in the output view 261 Configuring the display 263 Setting options in the system browser 264

21 The Window Browser 265

Introduction 265 Configuring the window browser 268 Performing operations on windows 270 Index 273

Preface

Conventions used in this manual

This manual assumes that you have at least a basic knowledge of Common Lisp. Many source code examples are used throughout the manual to illustrate important concepts, but only extensions to Common Lisp which are specific to the environment are explained in detail.

This manual *does* provide a complete description of the windowed development environment available in your Lisp image. This includes a description of the user interface itself, and a description of how the user interface interacts with Common Lisp.

Using the mouse

Throughout this manual, actions that you perform using the mouse are described in terms of the gesture used, rather than the combination of mouse buttons and keys that need to be used to perform the operation. This is because the buttons that are used are highly dependent on the platform you are running your Lisp image on, the operating system you are using, and even the type of mouse that you have attached to your computer. The mouse gestures available in the environment are described below.

Select

This is by far the most common mouse gesture, and is used for nearly all mouse operations in the environment. Use the select gesture to

- display a menu,
- choose a command from a menu which is already displayed,
- select items from a list or graph
- select or deselect a toggle switch,
- click on a button,
- position the mouse pointer in a piece of text.

Depending on the characteristics of your operating system or (if you are using a UNIX system) your window manager, you may also need to use select in order to move the mouse focus to another window.

If you are using a mouse with several buttons, you can nearly always select by clicking the left-most button, but you should refer to the documentation for your operating system or window manager if you are unsure. This is particularly true if you are using a mouse which has been set up for use by a left-handed person, since it is possible that the function of the mouse buttons has been reversed.

Multiple select

Multiple selection is used in lists and graphs when you want to select more than one item. You can select several items from any list or graph in the environment, and there are a large number of commands which can operate equally well on these multiple selections.

There are a number of standard ways of making multiple selections in a list or graph, depending on your operating system or window manager. Check the relevant documentation if you are unsure, or try any of the following:

- Holding down the Shift key while selecting an item.
- Holding down the Control key while selecting an item.
- The middle mouse button (if you have a three-button mouse).

Typically, in lists, holding down the Shift key lets you make a contiguous selection, and holding down the Control key lets you make a discontiguous selection.

- To select a block of items from a list, select the first item, hold down the Shift key, and then select the last item; the intervening items are also selected.
- To select several items which do not form a block, hold down the Control key while selecting each item individually.

This behavior is typical in a number of operating systems or window managers. You are probably familiar with it if you are familiar with using a mouse.

Double-click

The double-click gesture consists of two select gestures, performed in rapid succession. In general, any item in a list or graph may be double-clicked.

Double-clicking is usually a shortcut for selecting an item and choosing a common menu command, and the precise action that takes place depends on the context in which the double-click was performed.

Double-clicking can only be performed on single selections.

Alternate select

This is a less common gesture, and is used almost exclusively within Common LispWorks to display a popup menu.

If you are using a mouse with several buttons, you should find that you can perform this gesture by clicking the right-most mouse button. Refer to the documentation for your window manager or operating system if you are unsure.

Choosing menu commands

Throughout this manual, menu commands are shown in **This Bold Font**. In addition, submenus are indicated by use of the > character. Thus, for instance, the instruction

```
"Choose File > Open"
```

means that you should select the **File** menu on a menu bar, and choose the **Open** command in the menu that appears. Similarly,

```
"Choose Works > Tools > Editor"
```

means that you should display the **Works** menu by selecting it, select **Tools** from this menu to display a submenu, and choose the **Editor** command from this submenu.

Using the keyboard

Throughout this manual there are descriptions of commands that you can choose by typing at the keyboard. This is especially true when discussing the built-in editor, which relies heavily on the use of keyboard commands, and the Common Lisp listener, which uses many of the same commands.

Keyboard commands generally use a combination of alphanumeric characters together with the Control, Shift, Escape, and Meta keys.

In all cases, the Control, Shift, and Meta keys should be held down concurrently with the specified letter, whereas the Escape key should be pressed and released before pressing the specified letter. In the editor, Escape and Alt can be used interchangeably. This manual refers to Alt when referring to editor commands to reflect its ease of use.

Thus, for example:

Ctrl+s is read as "hold down the Control key and press S".

Ctrl+shift+A is read as "hold down the Control and Shift keys and press A".

ESC E is read as "press and release the Escape key, then press E".

Meta+E is read as "hold down the Alt key and press E" (remember that, in the editor, you could use Esc+E if you preferred).

For more information on using keyboard commands in the built-in editor and the listener, see Section 10.1.3 on page 111.

1

Introduction

This manual gives you a complete guide to the Common LispWorks development environment. This environment comprises a large number of windowbased tools which have been designed with the Common Lisp developer in mind. The following are among the features provided by the environment:

- A fully functional editor specifically designed to make writing Common Lisp source code as swift as possible.
- A Common Lisp listener for evaluating Common Lisp forms interactively.
- A range of other tools essential to the development process, such as a debugger, code profiler, and inspector.
- A wide range of browsers for examining different objects in your Lisp image, such as the generic functions or CLOS classes that have been defined.
- A tool for simplifying source code management; vital if you are involved in developing large applications.
- A tool for designing window-based interfaces to your applications. Apoint-and-click interface is used to design the interface itself, and all the necessary Common Lisp code is generated for you.

Because of the large number of tools available, consistency is a vital theme in the environment; each tool has a similar look and feel so that you need only spend a minimum amount of time learning how to use the environment.

In addition, there is a high degree of integration between the tools available. This means that it is possible to transfer pieces of information throughout the environment in a logical fashion; if you create an object in the listener, you can examine it by transferring it directly to the inspector. The class of objects that it belongs to can be examined by transferring it to a class browser, and from there, the generic functions which have methods defined on it can be browsed.

To reflect these themes of consistency and integration, the earlier chapters in this manual deal with the generic aspects of the environment, while at the same time introducing you to the more important tools.

1.1 Major tools

The environment supports a wide range of tools which can help you to develop and maintain Common Lisp source code more quickly and efficiently. This section gives you a brief introduction to the most important ones. You can create any of the tools described here by choosing the appropriate command from the **Tools** menu of the podium window.

For full details about any of these tools, see the relevant chapter. The second part of this manual covers each of the tools in the order that they are found on the **Tools** menu of the LispWorks podium.

1.1.1 The listener

A Common Lisp listener is provided to let you evaluate Common Lisp forms. This tool is invaluable as a method of testing your code without necessitating compilation or evaluation of whole files of Common Lisp source code.

1.1.2 The editor

A built-in editor is provided to allow you to develop Common Lisp code. It is based on EMACS, an editor which you may already be familiar with. The built-in editor offers a wide range of functions specifically designed to help you develop Common Lisp code, and it is fully integrated into the environment so that code being developed is immediately available for testing.

1.1.3 The class browser

This tool allows you to examine the Common Lisp classes that are defined in your environment. You can look at the superclasses and subclasses of a given class and see the relationships between them, and you can examine the slots available for each class.

In addition, you can examine the functions and methods defined on a given class, or the precedence list or initargs for the class.

1.1.4 The output browser

The output browser collects and displays all output from the environment which may be of use. This includes warning and error messages displayed during compilation and output generated by tracing or profiling functions. Many other tools in the environment also provide you with an output *view*, which lets you see any output which is appropriate to that tool.

1.1.5 The inspector

The inspector lets you examine and destructively modify the contents of Common Lisp objects. It is an invaluable tool during development, since it lets you inspect the state of any part of your data at any stage during execution. Thus, it is easy to see the value of a slot and, if necessary, alter its value, so that you can test out the effects of such an alteration before you make the changes necessary in the source code itself. Introduction

A Short Tutorial

This chapter gives you a short tutorial illustrating simple use of some of the major tools in the environment, and attempts to familiarize you with the way that tools can be used to good effect when developing Common Lisp applications.

Note that some of the examples given in this chapter use symbols taken from the CAPI library. Do not worry if you are not familiar with the CAPI. It is not essential that you fully understand the example code used in order to gain benefit from the tutorial. If you wish to learn more about the CAPI, you should refer to the *CAPI User Guide* and the *CAPI Reference Manual*, both of which are supplied with your software.

To maintain continuity, try to work your way through the whole of this tutorial in one session.

2.1 Starting the environment

To start Lisp under Windows 95, Windows 98 or Windows NT 4.0:

- 1. Click Start on the task bar.
- 2. From the Start menu, choose Programs > Harlequin > LispWorks.

After a short pause, you should see a copyright banner, followed closely by the LispWorks podium window and a listener. The LispWorks podium is shown in Figure 2.1.



Figure 2.1 LispWorks Podium

The podium window is automatically displayed whenever you start the environment. It contains a menu bar, which gives you access to various functions, as well as all the other tools in the environment, and a *button bar*, which gives you quick access to some of the more convenient menu commands.

Like many other applications, the menu bar contains File, Edit, Windows and Help menus. The Works menu contains commands that apply to the *active window*. The active window is listed in the LispWorks podium, underneath the button bar. The File menu contains file saving, compilation and printing options. The Tools menu gives you access to all of the Common LispWorks tools. The Debug menu is active when you are in the debugger and provide access to debugging commands. The Windows menu lists all the active windows you have running. The History menu contains a list of commands you have recently performed for that tool.

Note: If you wish to exit the Lisp image during this tutorial or at any other time, choose **File > Exit**.

2.2 Creating a listener

The listener is a tool that interactively evaluates the Lisp forms you enter. During a typical session, you evaluate pieces of code in the listener, then examine the effects in other tools, returning to the listener whenever you want to make more changes. The structure of this tutorial reflects this two-stage approach.

A listener is created when you start up LispWorks. This section of the tutorial demonstrates some of its more useful features. A listener is shown in Figure 2.2 below.



Figure 2.2 Listener

The listener contains two views: the listener view and the output view. At the bottom of the listener is an echo area that is visible in either view. The echo area is used to prompt you for information when performing editor com-

mands such as searching for text. You can switch between the two views by clicking the Listener and Output tabs respectively. You can evaluate Lisp forms in the listener view by typing the form, followed by Return. Any output that is produced is displayed in the Listener view.

1. Type the following form into the listener and press Return.

(+ 1 2)

The result of the evaluation, 3, appears in the listener, and a new prompt is printed. Notice that the number in the prompt has been incremented, indicating that a form has been evaluated.

Because you may want to enter a number of very similar forms, commands are provided which make this easy.

2. Press Meta+P.

The form that you just evaluated is printed at the new prompt. You can press Return to evaluate this form again, or, more usefully, you can edit the form slightly before evaluating it.

3. Press Ctrl+B to move the cursor back one space. Now press the Back-space key to delete the number 2, and type 3 in its place.

You have edited the form (+ 1 2) to create a new form, (+ 1 3).

4. Press Return to evaluate the new form.

The result of the evaluation, 4, appears in the listener, followed by another new prompt, with the prompt number incremented once again.

2.3 Using the debugger

A debugger tool is provided to help track down the cause of problems in your source code. This section introduces you to some of the ways in which it can be used.

1. Type the following definition into the listener:

```
(defun test ()
  (let ((total 0))
    (loop for i below 100 do
        (incf total i) when (= i 50) do
        (break "We've reached fifty"))))
```

This function counts from 0 to 100, accumulating the total as it progresses, and forces entry into the debugger when the count has reached 50.

2. Next, call the function by typing (test) into the listener.

Initially, the command line debugger is entered. This is a debugger which can be used from within the listener itself. More details about the command line debugger can be found in the relevant documentation for the Lisp image that you are running.

3. To enter the debugger tool at this point, choose **Debug > Listener > Start GUI Debugger** from the podium.

The debugger tool appears, as shown in Figure 2.3.

Error condition		Del	bugger backtra	ce
\backslash				
🛃 Debugger 3				_ 🗆 ×
<u>C</u> ondition <u>France</u> <u>Re</u> Condition:	starts		/	
We've reached fifty 🔪				A V
Backtrace:				
INVOKE-DEBUGGER BREAK SPECIAL::BYPASS-EVA	LHOOK	/		
SPECIAL::BYPASS-EVA CAPI::CAPI-TOP-LEVEL (SUBFUNCTION 2 (MET CAPI::INITIALIZE-AND- (SUBFUNCTION 1 CAPI (SUBFUNCTION 1 MP::	LHOOK -FUNCTION (HOD CAPI::(PROCESS-E\ I::CALL-AND- INITIALIZE-P	CREATE :AF /ENTS PROCESS-E ROCESS-ST	TER (CAPI:INTEF VENTS) ACK)	RACTIVE-PANE
•				
<u>V</u> ariables:				
Arg N V LOOP::IT T #: by-456 1 #: to-455 100 I 50 TOTAL 1275				
Report	Bug	Coptinue	Abort	
JReady.		\rightarrow		
}tate of variables for frame	selected		Control butto	ons

Figure 2.3 Debugger tool

The debugger tool gives a view of the backtrace (in the Backtrace pane), showing the functions that are on the stack, and their internal variables (including any arguments) at the point that the error occurred.

4. In the Backtrace pane, click on **TEST** if it is not already selected.

This displays, in the Variables list, the values of the variables in the example. Notice that the value for i is 50, as you would expect.

There is a row of buttons at the bottom of the debugger which let you perform a number of different actions.

5. Click Continue to exit the debugger.

The debugger disappears from the screen, and the command line debugger in the listener is exited, leaving you at the Lisp prompt in the listener.

2.4 Viewing output

There are many different ways to view output generated by the environment. In many tools, for example, output appears as soon as it is generated — this happens, for instance, when you compile code in the built-in editor.

At other times, you can view output in a tool called the output browser. This tools collects together *all* the output generated by the environment, and is particularly useful for viewing output generated by your own processes (which cannot be displayed in any other environment tool). The output browser displays all the output sent to the default value of the variable <code>*standard-out-put*</code>.

1. Evaluate the following in the listener.

```
(capi:contain
 (make-instance 'capi:push-button-panel
    :items '(:red :yellow :blue)
    :selection-callback
    #'(lambda (data interface)
        (format t
        "Pressed button in interface ~S~% data=~S~%"
        interface data))))
```

This is a piece of CAPI code that creates a window with three buttons, labeled **RED**, **YELLOW** and **BLUE**, as shown in Figure 2.4. Pressing any of these buttons returns the value of the button pressed.



Figure 2.4 Example CAPI window

- 2. Click on the Output tab in the listener.
- **3.** Try clicking on any of the buttons in the window you just created, and look at the output generated.
- **4.** Now try a second example by typing the form below into the listener at the current prompt (remember to click the Listener tab in the listener first).

The object that this code creates is going to demonstrate the inspector tool. The code above creates a window containing a text input pane. You can type text directly into a text input pane, and this can be passed, for instance, to other functions for further processing.

5. Type the word hello into the text input pane and press Return. Look at the generated output in the output view.

2.5 Inspecting objects using the inspector

The variables *, **, and *** hold the results of expressions which have been evaluated in the listener. * always holds the result of the last expression evaluated; ** holds the previous value of *, and *** holds the previous value of **. These variables (* in particular) are not only useful in their own right; the environment uses them to pass values between different tools.

1. Make sure the listener view is visible, and type *.

If you have followed this tutorial so far, the text input pane object that you created above is returned. This is because the capi:contain function returns the object that is being contained. You can easily inspect this object more closely in the inspector tool.

2. Choose Works > Values > Inspect from the podium menu bar.

This creates an inspector tool which displays the object currently contained in *, as shown in Figure 2.5.

Management of 1				
Text Input Pane Local Slots				
<u>F</u> ilter:	Matches: 36			
Attribute CAPI:ACCEPTS-FOCUS-P CAPI:BACKGROUND CAPI-INTERNALS:CALLBACK CAPI:CALLBACK-TYPE CAPI-INTERNALS:CARET-POSITION CAPI-INTERNALS:CHANGE-CALLBACK CAPI:CCHANGE-CALLBACK-TYPE CAPI:COLOR-REQUIREMENTS CAPI:CONFIRM-CHANGE-FUNCTION CAPI:CONFIRM-CHANGE-FUNCTION CAPI:CURSOR CAPI:DECORATION CAPI:DECORATION CAPI:FLAGS CAPI:FONT CAPI:FOREGROUND CAPI:GEOMETRY-CACHE CAPI:HINT_TABLE	Value T VIN32:COLOR_WINDOW #(LAMBDA (TEXT INTERFAC DATA-INTERFACE 0 NIL :FULL NIL SIL NIL NIL #S(CAPI::DECORATION-INFC T 0 #S(GRAPHICS-PORTS::FON1 WIN32:COLOR_WINDOWTE: # <capi::pane-geometry (chabacter="" (minawidth=""]<="" n="" t="" td=""></capi::pane-geometry>			
Ready.				
Object : # <capi:text-input-pane ""="" 2038606c=""></capi:text-input-pane>				

Figure 2.5 Examining a text input pane in the inspector

The commands in the **Works > Values** menu always act upon the current value of *. This enables you to pass a value easily from one tool to another. Similar menus can be found in all tools in the environment, and the behavior of each command is consistent throughout the environment. For a full description of the functionality of these menus, see Section 3.8 on page 31.

The main part of the inspector is a list of all the slots in the object being inspected. This list shows both the name of each slot and its current value. Above this list is an box labeled Filter. This lets you filter the information shown in the main list, which can be useful when you are inspecting objects with a large number of slots. The name of the object being inspected appears immediately below the menu bar.

3. Click in the Filter box, type the word text and then press Return.

This restricts the display in the inspector to only those items which contain the string "text", either in the slot name or in the slot value.

After using the filter, you can easily see that one of the available slots contains the text that you typed into the text input pane.

The inspector always displays the actual instantiation of a given object (as opposed to a copy of it), so that you can be certain that any changes to the object itself are reflected in the inspector.

4. Display the text input pane that you created earlier.

If you can no longer see it, choose **Windows > Container**; this is a simple way to display any of the windows and tools that you have created so far. (There are actually two windows with this name; if you choose the wrong one first of all, then just choose the other one.)

- 5. Click in the text input pane and delete the word hello. Type goodbye and press Return.
- 6. Select the inspector to make it the active window and choose Tools > Update.

The description of the text slot now reflects the new value you specified.

7. Close the inspector by clicking the Close button in its title bar.

You can close any window in the environment in this way, although there are often other ways of closing windows.

2.6 Examining classes in the class browser

This section shows you how to use the class browser tool to examine information about the Common Lisp class of any given object. The examples given use the text input pane object that you created earlier, and show you how you can programmatically change the values of a slot.

1. In the listener, type ***** once again.

Notice that the * variable still contains the value of the text input pane object. This means that it is easy to perform several actions on that object. Notice further that the environment is aware that the object has been changed: the value returned by * reflects the change to the text slot that you made in the last section.

2. From the LispWorks listener, choose Works > Values > Class.

This creates a class browser, shown in Figure 2.6, which allows you to examine the class of the object contained in *.



Figure 2.6 Examining the class of an object using the class browser

The class browser contains more areas than the inspector. In the Class box, the name of the current Common Lisp class is printed. The list below the Filter box displays the slots available to the current class, and list labeled Description displays the description of any selected slot. The filter works in the same way as the inspector's filter. There is also a checkbox labeled Include Inherited Slots. Clicking this checkbox lets you switch between displaying all the slots defined on the current class and all its superclasses, and only those slots

defined directly on the current class. By default, slots defined on any superclasses (inherited slots) are shown in the main area.

3. Filter the display as you did for the inspector; click in the Filter box, and this time type the word foreground and press Return.

Only those slots with the string "foreground" in their names are displayed.

4. Select the CAPI::FOREGROUND slot from the list. A description of the slot appears in the description area, including information such as the initargs, readers, and writers of the slot.

Notice that the text input pane has both a reader, capi:simple-pane-foreground, and a writer, (setf capi:simple-pane-foreground). We can use this information to programmatically change the text shown in the text input pane.

5. Type this form into the listener:

(setf (capi:simple-pane-foreground *) :red)

The text displayed in the text input pane is displayed in red to reflect the new value you have specified. Notice how you were able to use the * variable to refer directly to the text input pane object itself.

2.7 Summary

In this introductory tutorial you have seen how to perform the following actions:

- Start the windowing environment.
- Evaluate and re-evaluate Common Lisp forms using the listener.
- Invoke the debugger, follow the backtrace that it produces, and return from the error which caused entry to the debugger.
- Collect and display data generated by your own code in the output browser.
- Use the inspector to examine the current state of an object.

• Use the class browser to find out detailed information about a given class, so that you can make arbitrary programmatic changes to an instance of that class.

The next two chapters describe elements of the environment which are common to all tools.

Other chapters in this manual describe the other tools available in the environment. Each chapter is intended to be reasonably independent of the others, so you can look at them in any order you wish. You are advised to study the chapters on the basic tools, such as the inspector, the class browser and the editor first, since a knowledge of these tools is vital if you want to get the best out of the environment.

Common Features

The environment has been designed so that its features are consistent throughout, and tools have a uniform look and feel. All tools have certain characteristics which look the same, and behave in a consistent manner. By making as many common features as possible, learning how to use each tool is much simpler.

Chapter 2, "A Short Tutorial", introduced you to some of the major tools in the environment, demonstrating the commonality and high integration between them, and showing how this can be used to good effect in the development process. This chapter describes these common features in more detail.

Most of the common features in the environment can be found under the File, Edit, Works, History, Windows and Help menus in the LispWorks podium. Using the commands available under these menus you can:

- move to any other tool
- cut, copy or paste via the clipboard
- perform search and replace operations
- re-issue a previous command, or re-examine an object
- perform operations such as loading and saving files

The commands in these and all other menus in the podium operate on the active window, the name of which is displayed at the bottom of the LispWorks podium. The active window can be any window within the Common LispWorks environment: even the LispWorks podium itself. In addition, some other conventions have been adopted throughout the environment:

- Many tools have a number of different views: ways of displaying information. Each view is made available by clicking on a different tab in the tool.
- Lists displayed in many tools can be filtered in order to hide redundant or uninteresting information.

These features are described in full in this chapter. Please note that subsequent descriptions of individual tools in the environment do *not* include a description of these menus, unless a feature specific to the individual tool is described.

Online help is also available from the **Help** menu in any window. These facilities are described in Chapter 4, "Getting Help".

Many tools allow you to display information in the form of a graph. These graph views behave consistently throughout the environment, and a description of the graph features offered is given in Chapter 5, "Manipulating Graphs".

3.1 Loading and displaying tools

There are many tools available, and you can display them in a number of ways. You can also control how a given tool is re-used within the environment.

3.1.1 Displaying existing windows

Choose the **Windows** menu from the LispWorks podium to display a list of all the windows currently available in the environment. Choosing any item from this list brings the window to the front of the display.

3.1.2 Loading and displaying tools

To load or display any tool:

1. Choose the **Tools** menu from the LispWorks podium.

Every tool in the environment is listed in this menu.

2. Choose the tool you require from the menu.

The tool is loaded (if necessary), and displayed. Using this method can be very useful if you have several windows open, since you may not remember immediately whether a given tool is loaded or not.

3.1.3 Re-using windows

By default, the environment always re-uses an existing window if possible. So, if you choose **Tools > Editor**, an existing editor appears if one has already been loaded.

If you prefer, you can configure the environment so that a new tool is created every time you choose one of the options under the **Tools** menu. You can do only for specific tools, of for every tool in Common LispWorks.

To switch off the reuse of a specific tool:

1. Choose Tools > Options from the LispWorks podium. The options dialog appears.

Reuse all tools is one of the options displayed on this dialog. By default, this is selected, indicating that the active window is re-used.

2. Uncheck Reuse all tools to deselect the option.

The active windows are now no longer reusable. Next time you create a new instance of a tool from the **Tools** menu, a new instance of it is created.

3. Choose **Tools > Options** and check **Reuse all tools** again to switch the option back on.

Note: You can also control the reuse of tools from the **Tools > Customize > Reusable** menu in the LispWorks podium.

3.1.4 Creating menu bars

By default, the only window in Common LispWorks to contain a menu bar is the LispWorks podium. The menu commands in this menu bar operate on whichever Common LispWorks window is currently active. If you prefer each tool to have its own menu bar, you can configure this by choosing **Tools > Options** and toggle **Each tool has its own menu bar**.

Note: This manual assumes the default Common LispWorks configuration: only the LispWorks podium has a menu bar.

3.1.5 Copying windows

Choose Tools **> Clone** in a given tool or window to make a copy of that tool or window. This is useful, for instance, if you wish to have two different views on an object simultaneously, and allows you to have several copies of a tool or application without having to change its properties using **Tools > Customize > Reusable**.

3.1.6 Closing windows

Close any window in the environment using one of the following methods:

- Click the Close button at the top right of the window.
- Click in the icon at the top left of the window to display the windows control menu, and choose **Close**.
- Press Alt+F4.

3.1.7 Updating windows

To manually update any tool, choose **Tools > Update**.

Updating a tool is a useful way of making a snapshot of an aspect of the environment that you are interested in. For instance, imagine you want to compare a number of instances of a CLOS class against a known instance of the same class using the inspector. You can do this as follows:

1. Inspect the known instance using an inspector.

You might do this by creating the instance in a listener and then using **Works > Values > Inspect** to transfer the instance to the inspector.

- 1. Make sure the inspector is the active window, and choose Tools > Clone to make a copy of it.
- **2.** In the listener, create a new instance of the same class, transfer it to the inspector, and compare them to the original instance that is still displayed in the clone.

3.2 Setting global preferences

Chose **Tools > Options...** from the podium to invoke the global preferences dialog. This dialog is used to specify global preferences, such as the name of the initialization file to use.

🛃 LispWorks: Global Preferences	×
- Windows Options	
Each tool has its own menu bar.	
☑ <u>R</u> euse all tools.	
Initialization File	
C:\.lispworks	X 🔅
<u>0</u> K	<u>C</u> ancel

Figure 3.1 The Global Preferences dialog.

3.2.1 The windows options

Checking **Each tool has it's own menu bar** tells LispWorks to attach menu bars to all LispWorks tool windows, such as the editor, the class browser, the

inspector, and so on. If this is not checked then the podium is the only tool containing a menu bar.

Checking **Reuse all tools** ensures that LispWorks uses an existing tool rather than starting up a new copy. For example, if an editor is already open, choosing **File > Open** and selecting a new file causes the file to be opened in the existing editor.

3.2.2 Initialization file

The .lispworks file is the default file to be loaded automatically when LispWorks is started. You can edit it to contain Common Lisp code for evaluation to initialize the image to suit your needs.

The global preferences dialog can be used to specify a different initialization file. You can either enter the path and filename directly into the text input box, or use the path button to call up a file browser to select a file. Clicking on

undoes any alterations entered.

3.3 Quitting the environment

Choose File > Exit to exit Lisp completely. A dialog asks you to confirm your exit when you choose this command.

3.4 Performing editing functions

This section discusses commands available in the **Edit** menu while any window is active. These commands fall into five areas:

- Undoing changes.
- Using the clipboard.
- Selecting text and objects.
- Searching for text.
- Substituting text.
3.4.1 Undoing changes

You can undo changes made in a tool using **Edit > Undo**. This facility is most useful in the editor and listener— see Section 10.9 on page 122 for more details.

3.4.2 Using the clipboard

You can use the clipboard to transfer data between tools, or even between the environment and other applications that you are running. There are three commands available, as follows:

Choose **Edit > Copy** to put the selection or "primary object" onto the clipboard.

Choose **Edit > Cut** to put the selection or "primary object" onto the clipboard and remove it from the tool it was copied from.

Choose **Edit > Paste** to put the contents of the clipboard into the current tool.

Unlike the clipboard in many other applications, the Common LispWorks clipboard can contain the Common Lisp object itself. This makes the Common LispWorks clipboard an exceptionally powerful tool, allowing you to pass objects between different tools in the environment so that they can be examined in different ways.

Use of **Copy** or **Cut** followed by **Paste** lets you transfer items between tools, or to different parts of the same tool. There are several ways to use these commands:

- In the listener you can **Copy** the previous form to the clipboard and then **Paste** it into another tool. Because the Common Lisp object itself is copied to the clipboard, it is treated correctly—for instance, if you paste it into an inspector, it is inspected.
- In the editor, you can **Copy** chunks of text and **Paste** them into different places, either within the same file or between different files. If you have sections of code which are very similar, rather than typing each section out explicitly, just **Paste** in the same section as many times as you need and change only the relevant parts. Section 10.11 on page 123 describes a number of more sophisticated methods that can be used in the editor.

• Between any of the tools, you can **Copy**, **Cut**, and **Paste** Common Lisp objects. You can, for instance, make an instance of a class in the listener, **Copy** it, then **Paste** it into a class browser to examine its description.

As well as the menu commands, you can use the 👗, 🗈 and 🛍 buttons in the button bar of the LispWorks podium.

Note: You can also transfer data within the environment using the standard actions commands described in Section 3.8 on page 31.

3.4.3 Selecting text and objects

Choose Edit > Select All or Edit > Deselect All to select or deselect all the text in an editor or listener window, or all the items in a list or graph. These commands are useful whenever there is too much information to be able to select items one at a time.

3.4.4 Searching for text and objects

You can search for and change text in most tools using Edit > Find, Edit > Find Next, and Edit > Replace.

Choose **Edit > Find** to find an item in the current tool (this might be a piece of text, or a fragment of Common Lisp, or an object, depending on the tool). You must supply an item to find in the dialog that appears.

Choose **Edit > Find Next** if you want to search for the next occurrence of an item you have already found. This command does not prompt you for an item to find, and so is only available if you have already found something.

Choose **Edit > Replace** if you want to replace one string of text with another. The echo area of the active window prompts you for a text string to find, and a text string to replace it with. This command is only available in the editor and the listener, and is most useful in the editor.

3.5 The history list

The *history list* of a tool stores the most recent events which have been carried out in that tool, or the most recent objects which have been browsed in it.

The **History > Items** submenu provides a list of these events (or objects), allowing you to repeat any of them (or browse them again) by choosing them from the menu. This gives you an easy way of repeating forms in the listener, inspecting objects or browsing classes again, and so on.

The menu lists the last ten *unique* items to have entered the history list of the active window. Because each entry is unique, some items may have occurred more than ten events ago.

If the editor is the active window, the **History> Items** submenu lists the buffers currently open.

3.5.1 Repeating events from the history list

The easiest way of repeating an event from the history list is to choose it from the **History > Items** submenu. There may be times, though, when this is inconvenient (the items on the list may be too long to be able to distinguish between them easily, or you might want to repeat an item that occurred more than ten events ago). In such cases, there are three commands which offer an alternative way of choosing items.

Choose **History > Previous** to perform the previous item in the history list of the tool. This is usually the most recent event you have performed, but may not be (if, for instance, the last action was itself an event that was already on the history list).

Choose **History > Next** to perform the next item in the history list. This item is not usually available unless the last event you performed involved an item already on the history list.

Note: You can also use the *mathefamiliary* and *statements* buttons in the button bar on the Lisp-Works podium.

3.5.2 Editing the history list

Choose **History > Modify** to remove items from the **History > Items** menu. A dialog appears that contains all of the items in the current **History** menu. Select the items you wish to *retain*, and click **OK**. Any items which were *not* selected in the dialog are removed from the history list.

3.6 Operating on files

The **File** menu allows you to perform operations on files stored on disk. It is only available for tools which need to interact with the files you have stored on disk, such as the listener and editor.

The default commands available in the File menu are described below. Note that in some tools, the File menu contains additional commands specific to that tool. Please refer to the relevant chapters for each tool for a description of these additional commands.

Choose File > New to open a new buffer in the built-in editor. If an editor window has not yet been created, this command also creates one. The new buffer is unnamed. Alternatively, you can click the button in the button bar on the LispWorks podium, shown in Figure 2.1, page 6.

Choose File > Open to open an existing file in a new editor buffer. Where appropriate, a dialog appears, allowing you to choose a filename. If an editor window has not yet been created, this command creates one. Alternatively, you can click the \overrightarrow{r} button in the button bar on the LispWorks podium, shown in Figure 2.1, page 6.

Choose File > Load, File > Compile, and File > Compile and Load to load, compile, or compile *and* load a file of Lisp source code. When appropriate, each command displays a dialog, allowing you to choose the file you want to load or compile.

Choose File > Print to print a file. A dialog allows you to choose a file to print. The current printer can be changed or configured by using the standard Windows Control Panel.

Choose File > Browse Parent System to view the parent system of the current file in the system browser. This command is only available if the parent system has already been loaded into the environment. See Chapter 20, "The System Browser" for a complete description of the system browser.

Note: As described above, the behavior of each command can vary slightly according to which tool is active when the command is chosen. For instance, choosing **File > Print** in the editor prints out the displayed file, whereas choosing **File > Print** in the listener prompts you for a file to print.

3.7 Displaying packages

Symbols can be displayed either with their package information attached or not. In Common LispWorks, symbols are displayed with the package name attached by default.

For example, suppose you have created a package FOO which includes a symbol named bar and a symbol named baz. Suppose further that you created a new package FOO2, which used the FOO package. This can be done as shown below:

```
(defpackage foo (:use "COMMON-LISP"))
(defpackage foo2 (:use "FOO" "COMMON-LISP"))
```

Note that in defining both packages, the COMMON-LISP package has also been used. It is good practice to use this package, to ensure that commonly-used symbols are available.

When creating packages which use other packages, exported symbols can be called without having to refer to the package name.

To clarify this, let us go back to our example.



Figure 3.2 Two example packages

We have two packages: FOO and FOO2. FOO contains 2 functions, bar and baz. Suppose that the function bar has been declared as exported, whereas the function baz is not.

When you are in the package F002, you can refer to bar without using the package name (because F002 uses F00 and bar is exported), but you must still refer to the F00 package name for baz (because baz is not exported). Note also

that when you are in any package other than FOO or FOO2, you can refer to foo:bar, but you must still refer to foo:baz.

Package names are usually displayed alongside symbols in a list. Having a package entry on every line can be confusing, especially if the majority of items listed are from the same package. To hide the package names for the symbols in the active window:

1. Choose Tools > Preferences.

The Preferences dialog for the active window appears.

2. Uncheck Show Package Names in this dialog.

3.7.1 Specifying a package

If you are working in a particular package, you can adjust the current tool to display its symbols as you would refer to them from that package—that is, as the package sees them. This can make listings clearer and, more importantly, can show you which symbols have been exported from a package.

Doing this changes the process package of the tool. This means that both displayed symbols and symbols typed into the tool are assumed to be in the package specified. This can be useful in a browser, for example, if you intend to browse a number of different objects which come from the same package.

To change the process package for the active window:

- 1. Choose Tools > Preferences.
- **2.** Delete the package name in the Package text field, and type in the name of the new package.
- **3.** Click **s** to confirm this new name.
- 4. Click Apply or OK to make the change.

Note: If you wish, you can partially type the package name and, then produce a list of possible completions by clicking **P**.

As an example, imagine you are looking at a list of symbols in the inspector. You are working in the package FOO, and some of the symbols in the inspector are in that package, while others are in another package. To change the current package of the inspector to **FOO**, follow the instructions below:

1. Choose Tools > Preferences.

The Preferences dialog indicates that COMMON-LISP-USER is the current package in this window.

- 2. In the Package field, delete the string COMMON-LISP-USER, and type FOO.
- **3.** Click **OK** to make the change.

In the inspector all the symbols available from FOO appear without the package prefix FOO. Similarly, all exported symbols in packages which FOO uses appear without a package prefix, while all others have an appropriate package prefix.

3.8 Performing operations on selected objects

In any tool, there are a number of operations that you can always perform on the selected objects, irrespective of the type of objects you have selected. This allows you to perform some powerful operations and also ensures a consistent feel to every tool in the environment.

In this context the term "selected objects" is meant in the widest sense, and can refer to *any* items selected *anywhere* in a tool, be it in a list of items, or a graph. It can also refer to the tool's *current object*: that is, the object which is currently being examined.

These operations are available throughout the environment, and are referred to as standard action commands. As with other commands that are specific to the active window, standard action commands are usually available from menus on the main menu bar of the tool you are using. The objects which are operated on by a given standard action command depend on the menu from which you chose the command.

As a more concrete example, consider examining the contents of Common Lisp objects using the inspector.

The standard action commands for the inspector are present in two places: the **Works > Object** menu, and the **Works > Slots** menu.

- Choose a standard action command from the **Works > Object** menu to perform an operation on the inspector's current object.
- Choose a standard action command from the **Works > Slots** menu to perform an operation on the selected components of the Common Lisp object.

Notice that in the first case, the object operated on is the tool's current object: you do not have to take any further action before performing the operation.

In the second case, the objects examined represent more specific pieces of information: you need to select them before you can perform the operation. This, therefore, examines more discrete pieces of information about the current object.

In general, any tool has at least two submenus like those described above. The first operates on the current object. What that object is, and hence the name of the submenu in which the commands are to be found, depends on the tool you are using. For instance, if you are examining classes, the commands can be found in a **Works > Classes** menu. If you are examining methods, they can be found in a **Works > Methods** menu.

Some tools may contain more than two such menus; precise details are given in the relevant chapters.

As a guide, if a menu has a plural for a name, the commands in that menu can be performed on multiple selections. If the menu name is not pluralized, commands only affect a single selection.

3.8.1 Operations available

The standard action commands available are described below. In these descriptions, the term "current object" refers to the Lisp object that is being acted upon by the menu command. This depends on the tool being used and the menu in which the command appears, but should be obvious from the context.

Choose **Browse** to browse the current object using an appropriate browser. A browser is a tool which lets you examine a particular type of Common Lisp object, and there are a large number of them available in the environment. Some of the browsers available are:

- the class browser, which lets you examine CLOS classes
- the generic function browser, which lets you examine the generic functions in the environment, and the methods you have defined on them

See the appropriate chapters for a full description of each browser; there is a chapter of this manual devoted each to available browser. The precise name of the **Browse** menu command reflects the type of browser that is used to examine the selected object. Thus, if the command is **Browse – Generic Function**, a generic function browser is used.

Choose **Class** to look at the class of the current object in a class browser. See Chapter 7, "The Class Browser" for full details about this tool.

Choose **Copy** to copy the current object to the clipboard, thus making it available for use elsewhere in the environment. Note that performing this operation on the object currently being examined by the tool (for example, choosing the command from the **Object** menu when an inspector is the active window) has the same effect as choosing **Edit > Copy**, whereas choosing this option from other menus (such as a **Description** menu) copies more discrete information to the clipboard.

Choose **Documentation** to display the Common Lisp documentation for the current object, if any exists. It is printed in a help window.

Choose **Find Source** to search for the source code definition of the current object. If it is found, the file is displayed in the editor: the cursor is placed at the start of the definition. See Chapter 10, "The Editor" for an introduction to the editor.

You can find only the definitions of objects you have defined yourself (those for which you have written source code)—not those provided by the environment or the LispWorks implementation.

Choose **Inspect** to invoke an inspector on the current object. See Chapter 13, "The Inspector", for details about the inspector. If you are ever in any doubt about which object is operated on by a standard action command, choose this command.

Choose Listen to paste the current object into the listener. Chapter 16, "The Listener" provides you with full details about this tool.

Choose **Function Calls** to describe the current object in a function call browser. See Chapter 11, "The Function Call Browser" for more details.

Choose **Generic Function** to describe the current object in a generic function browser. See Chapter 12, "The Generic Function Browser" for more details.

3.9 Using different views

Every tool in the environment has several different views, each of which can display information which is pertinent to the task at hand. You can switch to any of the available views by clicking on the appropriate tab at the top of the tool. When choosing a different view, the layout of the tool itself changes.



Figure 3.3 Clicking on tabs to display different views of a tool

In tools which are browsers, different views allow you to display different pieces of information about the same objects; for instance, in the class browser you can switch from a view which shows you information about the slots in a given Common Lisp class to one which shows information about the initargs of the class.

In other tools, different views may show you completely different types of related information. For example, in the listener you can switch from the listener view to a view that shows you any output that has been generated by the listener.

All tools have a default view when you first start them. The default view is the one which you are most likely to make most use of, or the one which you use first. When you first start the built-in editor, the default view is the text view. When you start a class browser, the default view shows you the slots available for the current class, as you have already seen.

3.9.1 Sorting items in views

You can sort the items displayed in the main area of any view using the Preferences dialog for a given tool. In tools where items can be sorted in this way, there are at least the following three commands available:

1. For any tool, choose Tools > Preferences.



Figure 3.4 Example Preferences dialog

Notice that some Preference dialogs, such as the one shown above, have several tabs. In these cases, the options described in this section are always available in the General tab.

The left hand side of the Preferences dialog is labeled Sort.

2. Choose one of the options in this area of the dialog to control the sort order of items in the active window.

The options available vary according to the tool, but at least the following will be available:

By Name	Sorts symbols in a list or graph according to the name of each item. The packages that the symbols are resi- dent in are ignored when this option is used; thus, the symbol vv:allocate would be listed before aa:vec- torize.
By Package	Sorts symbols in a list or graph according to the pack- age they are listed in. Thus, all symbols in the aa pack- age would be listed together, as would all symbols in the vv package. In addition, the aa package would be listed before the vv package. Within a given package, objects are listed in alphabetical order when using this option: thus, aa:carry-out-conditions would be listed before aa:vectorize.
Unsorted	Lists all symbols in a graph or list in the order in which they are defined in the source code. This can sometimes be a useful option in itself, and is always the quickest option available. You may sometimes want to use this option if you are displaying a large number of items and you are not filtering those items in any way.

Note: There are often other sort options available in the Sort area of the Preferences dialog, depending on the nature of the tool itself. These options are described in the chapter specific to each tool.

Only those views whose main area consists of a list or a graph can be sorted. In particular, the default views of tools such as the listener or the editor, which is an editor window which you can type directly into, cannot be sorted.

3.10 Tracing symbols from tools

For some tools, submenus under the some relevant main menues (for example, the **Works > Function** menu on the function browser) contain a **Trace** submenu that allows you to set tracing options for the selected function, method, macro, or generic function. This is a useful shortcut to the trace macro, since it gives you some control over tracing in the environment without having to work directly at the Common Lisp prompt.

A Trace submenu generally has the following commands:

Choose Trace to trace the currently selected function.

Choose **Trace Inside** to trace the currently selected function within the current context. Choosing this command sets the **:inside** option.

Choose **Trace with Break** to trace the currently selected function, and enter the debugger on entry to it. Choosing this command sets the **:break** option to t.

Choose Untrace to turn off tracing on the currently selected function.

Choose **Untrace All** to turn off tracing on currently traced functions. Note that this does not turn off tracing in the environment as a whole.

Choose **Toggle Tracing** to turn all tracing commands in the environment on or off.

3.11 Linking tools together

You can link together pairs of tools, so that changing the information displayed in one tool automatically updates the other. This can be done for virtually any tool in Common LispWorks, and provides a simple way for you to browse information and see how the state of the Lisp environment changes as you run your code. For instance, you can make a link from the inspector to the class browser so that every time you choose a class in the class browser, it is automatically inspected. Linking an editor window to the class browser is a good way of studying the implementation and design of a series of classes.

You can also link two copies of the same tool. This can be a very useful way of seeing two views of a tool at once. For instance, you could create a copy of the class browser by choosing **Tools > Clone**, and then link them together. By keeping one browser in the subclasses view, and the other in the slots view, you can automatically see both the subclasses *and* the available slots for a given class.

Linked tools have a master-slave relationship. One tool (the slave) gets updated automatically, and the other tool (the master) controls the linking process. To link together any two tools:

- 1. Select the tool that the link is to be established to. For example, to form a link from an inspector to a class browser to ensure that a class selected in the class browser is automatically inspected, you would user the Edit menu of the class browser.
- **2.** Choose **Edit** > Link > *fromtool* where *fromtool* is the title of the tool you wish to link from.

To break a link, select -- No Link -- instead of a specific tool.

To view all the current links that have been established, choose Edit > Link from > Browse Links... Select any links listed and click on Remove Link(s) remove them.

3.12 Filtering information

Many tools have views which display information of some sort in the form of a list. Items in these lists may be selected, and you can usually perform operations on selected items (for instance, by means of the standard action commands, as described in Section 3.8 on page 31).

Such lists are often very long, and there may be information displayed which you are not interested in. For instance, Common Lisp objects may contain a large number of slots, most of which are of no importance to your work.

Very often a list is accompanied by a Filter box which lets you hide such redundant information. Filter boxes consist of a pane into which you can type a string of text, as shown below, and are always positioned immediately above the list on which the Filter box operates.



Figure 3.5 Example Filter box

To use a Filter box, type in a text string and press Return, or click the Confirm button. Only those items that contain the specified string are displayed in the list—all the others are hidden from the display. The number of items that are listed is printed in the Matches area to the right of the Filter box.

To display all the items in a list once again, delete the string in the Filter box and press Return, or click the Cancel button.

3.13 Examining the window itself

You can examine either the CAPI representation of any window, or the underlying window itself, using the **Tools > Interface** menu.

This menu contains the standard action commands described in Section 3.8 on page 31. Thus, choose **Tools > Interface > Browse** to browse the CAPI representation of any window in a window browser, or choose **Tools > Interface > Inspect** to browse the underlying library object in a window browser.

Similarly, you can browse the Common Lisp classes of either the CAPI representation or the library model, or you can inspect their values, or paste them into a listener. For information about the tools mentioned, see Chapter 7, "The Class Browser", Chapter 13, "The Inspector" and Chapter 21, "The Window Browser".

Getting Help

All tools contain a **Help** menu which gives you access to a variety of forms of online help. This chapter describes how to use this online help.

4.1 Online manuals in HTML format

A complete documentation set is provided in the standard distribution, in the form of HTML files. These files can be found in the LispWorks distribution directory <*lwdir*>\1ib\4-1-0-0\manual\online\web\, where <*lwdir*> is your LispWorks installation directory, as described in the *Release and Installation Notes*. HTML is the SGML-derived markup language which has become the standard format for publishing information on the World-Wide Web. A wide variety of HTML browsers are commonly available, and you can use your preferred browser to view the online documentation.

Common LispWorks can link directly to the HTML files provided, allowing you to go straight to the most relevant documentation for the current context.

The documentation set was generated using Harlequin WebMaker. No proprietary extensions to HTML have been used, so you can use any HTML browser you want, although the Common LispWorks help menu only drives Netscape and Microsoft Internet Explorer.

4.1.1 Browsing manuals online

Choose **Help > Manuals** to browse any of the available manuals online. A dialog appears, allowing you to choose a manual to browse.

If you already have an HTML browser running, a link to the first page of the manual you choose is displayed in it. If you do not have a browser running, one is started for you.

4.1.2 Searching the online manuals

Choose **Help > Search**... to search the online documentation. The Search dialog, shown in Figure 4.1, appears.

Enter string to search for here.	Select other options here.
₩ Search	×
1 <u>S</u> earch for: button-panel	
 How would you like to search for the specifie Partial Search Whole Word 	ad text?
3 Search using: ○ Index ⊙ Contents	
<u>M</u> anuals ANSI Common Lisp Standard CAPI Reference Manual Common LispWorks User Guide Editor User Guide Guide to the CAPI	Packages CAPI-INTERNALS CAPI-LAYOUT COMMON-LISP EDITOR LISPWORKS-TOOLS
	<u>O</u> K Ca <u>n</u> cel

Select manuals to search here.

Select packages to search here.

Figure 4.1 Search dialog

This dialog lets you specify what you want to search for, and which manuals you want to search in.

Enter a string of text in the Search for area.

There are a number of additional options that you can set if you want:

• Select Whole Word if you want to confine your search to whole words only. Select Partial Search if you want to match part of a word as well. By default, partial searches are performed.

For example, if **Whole Word** is selected, searching for "pane" only matches the word "pane". If **Partial Search** is selected, searching for "pane" also matches "panels".

• You can choose whether to search the index or the table of contents of any given manual; select **Index** or **Contents** as appropriate. By default, indexes are searched, as these tend to produce the richest information.

Select the manuals you want to search in the Manuals list. If nothing is selected, all manuals are searched. You can select any number of items in this list.

Select the packages you want to search from the Packages list. If nothing is selected (the default), all packages are searched. You can select any number of items in this list.

Note that selections made in the Manuals and Packages lists reflect each other. If you choose one or more manuals, the relevant packages are also selected, and if you choose one or more packages, the relevant manuals are selected.

Once you have specified the search options, click **OK**. The results of the search are displayed in your HTML browser.

4.1.3 Getting help on the current tool

Choose **Help > On Tool** to get help on the current tool. This takes you to the appropriate online chapter of this manual.

4.1.4 Getting help on the current symbol

Choose **Help > On Symbol** to search for help on the symbol under the point (in an editor-based window) or the current object of a tool. This option displays

the Search dialog described in Section 4.1.2, but with options pre-selected to enable you to search for documentation on the current symbol. Click **OK**, and the results of the search are displayed in your HTML browser.

4.2 Online help for editor commands

You can display online help for any available editor command using the commands under **Help > Editing**. See Section 10.14 on page 134 for details.

4.3 Browsing manuals online using Adobe Acrobat

A complete documentation set is also provided in PDF (Portable Document Format). These can be found on the CD in the directory Doc\lib\4-1-0-0\manual\offline\pdf\. The installer creates links to these files in the Printable Documentation menu accessible from the Windows Start menu. You can view these files online and print them using Adobe Acrobat Reader Version 3.0 or above, which can be downloaded freely from the Adobe web site at http://www.adobe.com/.

Manipulating Graphs

Views that use graphs are provided in the class browser, function call browser, and window browser. These views let you, for instance, produce a graph of all the subclasses or superclasses of a given class, or the layouts of a given CAPI interface.

In the class browser, the subclasses and superclasses views use graphs. The function call browser uses graph views for its Called By and Calls Into views. There is only one view in the window browser, and that uses a graph.

All graphs can be manipulated in the same way. This chapter gives you a complete description of the features available.

All graphs have an associated graph layout menu, available by displaying a popup menu over the graph itself by using the alternate select gesture. This menu contains all the commands that are directly relevant to graphs.

5.1 An overview of the graph view

An example graph is shown in Figure 5.1 below. All graphs are laid out by Common LispWorks, so that their elements are displayed in an intuitive and easily visible hierarchy. A graph consists of a number of *nodes*, linked together by *branches*. By default, graphs in the environment are plotted from left to right: for any given node, the node to which it is linked on the left is known as its *parent*, and the nodes to which it is linked on the right are known as its *children*. The originating node of the graph (on the far left) is referred to as the *root node*, and the outer most nodes of the graph (on the far right) are referred to as *leaf nodes*. The root node does not have a parent, and leaf nodes do not have any children.



Figure 5.1 Example graph pane

You can select nodes in a graph pane in exactly the same way that you select items in a list. Similarly, you can copy nodes from a graph onto the clipboard in a manner consistent with use of the clipboard in the rest of the environment. When you copy any selected node onto the clipboard, the Lisp object itself is copied onto the clipboard, so that it can be transferred into other Common LispWorks tools.

5.2 Searching graphs

Sometimes graphs can be too large to fit onto the screen at once. In this case, it is useful to be able to search the graph for any nodes you are interested in. There are two commands which let you do this.

Choose **Edit > Find** to find any node in the graph whose name contains a given string. Choose **Edit > Find Next** to find the next node in the graph that contains that string. Whenever a matching node is found, it is selected in the graph. If necessary, the window scrolls so that the selected node is visible.

Note that you do not have to specify a complete node name: to find all nodes that include the word "debug" in their name, just type debug into the dialog. All searches are case insensitive.

A full description of these commands can be found in Section 3.4.4 on page 26.

5.3 Expanding and collapsing graphs

You may often find that you are only interested in certain nodes of a graph. Other nodes may be of no interest and it is useful, especially in large graphs, to be able to remove their children from the display. You can do this in a number of ways.

To collapse or expand any node in a graph, double-click on it. Thus, doubleclick on a leaf node to display its children (if it has any), and double-click on a non-leaf node to hide its children.



Figure 5.2 Expanding and collapsing nodes

For instance, in Figure 5.2, double-click on A to display B, C and D. Doubleclick on A once again to hide them. A small arrow is displayed next to any nodes which has any hidden children.

You can also collapse or expand nodes using the popup menu:

Choose **Expand Nodes** to expand the selected node. Choose **Collapse Nodes** to collapse the selected node.

5.4 Moving nodes in graphs

Although the layout of any graph is calculated automatically, you can move any node in a graph manually. This can be useful if the information in the graph is dense enough that some nodes are overlapping others.

To move the selected node, hold down the Shift key and select and drag the node to the desired location.



Figure 5.3 Moving a node in a graph

5.5 Displaying plans of graphs

Many graphs are too large to be able to display in their entirety on the screen. As with any other window, you can use the scroll bars to display hidden parts of the graph. However, you can also display a plan view of the entire graph.

To display the plan view of any graph, hold down the Control key and select the graph, or choose **Enter Plan Mode** from the popup menu. The graph is replaced by its plan view, similar to the one shown in Figure 5.4.



Figure 5.4 Example plan view

Each node in the original graph is represented by a rectangle in the plan view. The currently selected node is shown as a filled rectangle, and all other nodes are clear. You can select nodes in the plan view, just as you can in the normal view.

A dotted grid is drawn over the plan view; you can use this grid to alter the section of the graph that is shown in the normal view. The size and position of the grid represents the portion of the graph that is currently displayed in the normal view.

• To move the grid, so that a different part of the graph is shown in the normal view, hold down the Shift key and select and drag the innermost rectangle of the grid. The entire grid moves with the mouse pointer.

• To resize the grid, so that a different proportion of the graph is shown, hold down the Shift key and select and drag the outermost rectangle of the grid. The entire grid will resize. You can select any part of the grid except the innermost rectangle to perform this action.

To return to the normal view, hold down the Control key and select the graph again, or choose **Exit Plan Mode** from the popup menu. The part of the graph indicated by the grid in the plan view is displayed.

5.6 Preferences for graph layouts

A number of graph layout preferences can be set for any tool that uses graphs. These preference settings are available in the Preferences dialog for the tool, and can be displayed by doing either of the following:

- Choose **Preferences** from the graph layout popup menu.
- Choose **Tools > Preferences** from the relevant tool, and click on a graph layout tab in the Preferences dialog.

For example, the graph layout preferences for the class browser are shown in Figure 5.5.

🛃 Class Browser 1	Preferences X		
General Graph Layo Layout C Left to Right O Right to Left O Top Down O Bottom Up	Slots/Functions Max. Expansion Depth 2 Breadth None Plan Mode Rotation		
<u>Q</u> K <u>C</u> ancel <u>Apply</u>			

Figure 5.5 Graph layout preferences

This section describes the options available in the graph layout tabs of the Preference dialogs for any tool that uses graphs.

5.6.1 Altering the depth and breadth of graphs

For large graphs, you may find that you want to alter the maximum depth and breadth in order to simplify the information shown. Each graph has its own depth and breadth setting, which is used for all graphs drawn in that tool. These are available in the **Max Expansion** panel of the graph layout tab in the Preferences dialog.

The depth and breadth of a graph are depicted in Figure 5.6.



Figure 5.6 Depth and breadth of graphs

Choose a number from the Depth list to change the maximum depth of graphs in a given tool. The depth of a graph is the number of generations of node which are displayed. Most graphs have a default initial depth of 2, which means that you must expand any nodes you want to investigate by doubleclicking on them yourself. The default value is 2.

Note that the maximum depth setting is ignored for nodes which you have expanded or collapsed. See Section 5.3 on page 47.

Choose a number from the Breadth list to change the maximum breadth of a given tool. The breadth of a graph is the number of child nodes which are displayed for each parent. If there are more children than can be displayed (the maximum breadth setting is less than the number of children for a given node) an extra node is visible. This node is labeled "…", followed by the number of nodes that remain undisplayed. Double-clicking on this node expands it, allowing you to display the additional children without having to alter the maximum breadth setting for the whole graph. By default, the maximum breadth is set to None, so that all the children for a node are displayed, no matter how many there are. An example of this feature is shown in Figure 5.7 below.



Figure 5.7 Displaying children hidden by the maximum breadth setting

To ensure that all available information is graphed in a given tool, set both the maximum depth and maximum breadth to None.

5.6.2 Displaying different graph layouts

As already mentioned, graphs are laid out from left to right by default, but they can be laid out in any orientation you want. This can be configured in the Layout panel of the graph layout tab in the Preferences dialog.

Click "Left to Right" to layout a graph from the left of the screen to the right, as shown in Figure 5.8. This is the default orientation for every graph in the environment.



Figure 5.8 Left to right layout

Click "Right to Left" to layout a graph from the right of the screen to the left, as shown in Figure 5.9.



Figure 5.9 Right to left layout

Click "Top Down" to layout a graph from the top of the screen to the bottom, as shown in Figure 5.10.



Figure 5.10 Top down layout

Click "Bottom Up" to layout a graph from the bottom of the screen to the top, as shown in Figure 5.11.



Figure 5.11 Bottom up layout

At any time, you can choose **Force Re-layout of Graph** from the popup menu to force the graph to redisplay using the current layout method.

Manipulating Graphs

The LispWorks podium

When you start the environment, by default a window known as the *Lisp-Works podium* appears.



Figure 6.1 LispWorks podium

The LispWorks podium contains a menu bar, a button bar, a message area and a display area that shows the active window; that is, the window on which commands chosen from the **Works** menu will have effect.

The LispWorks tools have most if not all of these menu items in common with the podium.

The menu bar contains eight menus:

- The File, Edit and History menus contains commands described in Chapter 3, "Common Features".
- The **Tools** menu contains commands to create and configure Common LispWorks tools.
- The **Works** menu contains commands that operate on the active window.
- The **Debug** menu contains commands for debugging and is active only when you are in the debugger described in Chapter 9, "The Debugger Tool".
- The **Windows** menu lists all the current windows in the environment. To make any window the active window, choose it from this menu.
- The **Help** menu contains commands described in Chapter 4, "Getting Help".

Users already familiar with Windows will find that the **File** menu contain command available in similar menus in other applications.

The button bar provides quick access to some of the more common commands in the menus. Figure 6.1 shows each button, together with the menu command it represents.



Figure 6.2 LispWorks podium buttons and their functions

The Class Browser

The class browser allows you to examine Common Lisp classes. It contains seven views, allowing you to view class information in a number of different ways. You can display each view by clicking the appropriate tab. The available views are as follows:

- The slots view is used to look at the slots available to the class browsed. This view is rich in information, showing you details about items such as the readers and writers of the selected slot.
- The subclasses view produces a graph of the subclasses of the current class, giving you an easy way to see the relationship between different classes in the environment.
- The superclasses view produces a graph of the superclasses of the current class, giving you an easy way to see the relationship between different classes in the environment.
- The hierarchy view lets you see the superclasses and subclasses of the current class. It shows the immediate subclasses and superclasses using a text-based interface.
- The initargs view allows you to see the initargs of the current class together with information about each initarg. See Section 7.6 on page 81 for more details on how you can use this view.

- The functions view allows you to see information about the methods and generic functions that have been defined on the current class. See Section 7.5 on page 78 for details on using the information in this view.
- The precedence view is used to show the class precedence list for the current class. See Section 7.7 on page 83 for more details on how you can use this information.

To create a class browser, choose **Tools > Class Browser** from the podium. Alternatively, use Meta-x Describe Class from an editor or choose **Class** from any submenu that provides the standard action commands to invoke a class browser on the Lisp object referred to by that submenu. This automatically browses the class of the Lisp object. For more information on how the standard action commands refer to objects in the environment, see Section 3.8 on page 31.

7.1 Simple use of the class browser

This section describes some of the basic ways in which you can use the class browser by giving some examples. If you wish, you can skip this section and look at the descriptions of each individual view: these start with Section 7.2 on page 67.

When examining a class, the slot names of the class are displayed by default.

To examine a class, follow the instructions below:

1. In the listener, create a push button panel by typing the following:

```
(capi:contain
(make-instance 'capi:push-button-panel
                                :title "Test Buttons"
                               :items '(:one :two :three)))
```

The push button panel appears on your screen.

2. With the listener as the active window, choose Works > Values > Class.

This invokes the class browser on the button panel. The capi:push-button-panel class is then described in the class browser.


Figure 7.1 Examining classes in the class browser

Notice that, although you invoked the browser on an object that is an instance of a class, the class itself is described in the class browser. Similarly, if you had pasted the object into an inspector, the instance of that object would be inspected. Using the environment, it is very easy to pass Common Lisp objects between different tools in this intelligent fashion. This behavior is achieved using the Common LispWorks clipboard; see Section 3.4.2 on page 25 for details.

See Section 3.8 on page 31 for a full description of the standard action commands available.

7.1.1 Examining slots

A list of the slots in the current class is printed in the Slots area. By selecting any slot, you can examine it in more detail in the Description area.

While still examining the capi:push-button-panel class, select any slot in the Slots area.

From Classes:	CAPI:CALLBACKS
Slot Name:	CAPI::ACTION-CALLBACK
Type:	Т
Initargs:	:ACTION-CALLBACK
Initform:	NIL
Readers:	CAPI:CALLBACKS-ACTION-CALLBACK
Writers:	(SETF CAPI:CALLBACKS-ACTION-CALLBACK)
Allocation:	: INSTANCE

Figure 7.2 Description of a slot

A description of the slot is given in the Description area. For details about the information contained in this description, see Section 7.2.4 on page 70.

7.1.2 Examining inherited slots

By default, inherited slots (those slots which are defined in a superclass of the current class, rather than the current class itself) are listed in the Slots area along with the slots defined in the current class. Deselect the **Include Inherited Slots** button just above the Filter box to inhibit this listing.

1. While still examining the capi:push-button-panel class, click Include Inherited Slots.

No slots are displayed in the Slots area. This is because all the slots available to the capi:push-button-panel class are inherited from its superclasses. No slots are defined explicitly on the capi:push-button-panel class.

2. Select **Include Inherited Slots** again, and then select a few slots in the Slot area in turn.

Notice that the slot description for each slot tells you which superclass the slot is defined on.

7.1.3 Filtering slot information

The Filter box can be used to filter out information about slots you are not interested in. This is especially useful if you are examining classes which contain a large number of slots.

The example below shows you how to create an instance of a CAPI object, and then limit the display in the class browser so that the only slots displayed are those you are interested in:

1. In a listener, create a button object by typing the following:

```
(capi:contain (make-instance 'capi:list-panel
                                :items '("Apple" "Orange" "Pear")))
```

This creates a list panel object and displays it on your screen.

- 2. Choose Works > Values > Class in the listener to examine the class of the object in the class browser.
- **3.** Click the Initargs tab in the class browser to switch to the initargs view. Suppose you are only interested in seeing the callbacks that can be defined in a list panel.
- 4. Type callback in the Filter box and press Return.

🖉 Class Browser 1
Hierarchy Subclasses Superclasses Slots Initargs Functions Precedence
✓ Include Inherited Slots
Filter Callback Matches: 8
CAPI::ACTION-CALLBACK
CAPI::CALLBACK-TYPE
CAPI::EXTEND-CALLBACK
CAPI::HELP-CALLBACK
CAPI::ITEMS-CALLBACK
CAPI::RETRACT-CALLBACK
CAPI: SURVED-CALEDAUK
CAPI: SELECTION-CALIDACK
Description:
From Classes: CAPI:CALLBACKS
Slot Name: CAPI::ACTION-CALLBACK
Type: T
Initargs: :ACTION-CALLBACK
Initform: NIL
Readers: CAPI:CALLBACKS-ACTION-CALLBACK
Writers: (SETF CAPI:CALLBACKS-ACTION-CALLBACK)
Allocation: :INSTANCE

Figure 7.3 Using filters to limit the display in the class browser

You can immediately see the types of callback that are available to CAPI list panel objects. See the *CAPI Reference Manual* for details about these callbacks.

7.1.4 Examining other classes

There are two ways that you can examine other classes. The first is to type the name of the class you wish to see into the Class text box at the top of the browser. Press Return or click \checkmark and the named class is described.

1. While still examining class capi:list-panel, type capi:push-button-panel into the Class area.

The class capi:push-button-panel is described.

Because some class names may be potentially quite long, you can use *completion* to help save typing. If you press number when you have partially specified the name of a class in the Class text box, the environment attempts to complete what you have typed. If it cannot complete the class name, a dialog appears that lists all the possible alternatives. Double-click on any of these alternatives to place it in the Class text box.

The second way to examine other classes is by using the Superclasses and Subclasses lists available in the hierarchy view. Click on the Hierarchy tab to display the hierarchy view.

The main part of the hierarchy view consists of two lists:

- The Superclasses list shows all the superclasses of the current class.
- The Subclasses list shows all the subclasses of the current class.

Double-click on any superclass or subclass of the current class to examine it.

- 1. Double-click on CAPI:BUTTON-PANEL in the Superclasses list. The capi:button-panel class is described.
- 2. Double-click on CAPI: PUSH-BUTTON-PANEL in the Subclasses list. The capi:push-button-panel class is described again.

So, using the text view, you can easily look through the related classes in a system.

7.1.5 Sorting information

As with many of the other tools in Common LispWorks, you can sort the items in any of the lists or graphs of the class browser using the Preferences dialog. Choose **Tools > Preferences** in the class browser to display this dialog.

🛃 Class Browser 1 Preferences
General Graph Layout Slots/Functions Sort O Unsorted O By Name COMMON-LISP-USER O By Package Image: Common state
<u>□</u> K <u>C</u> ancel <u>Apply</u>

Figure 7.4 Setting class browser preferences

Under the General tab, there are three options for sorting items, listed in the Sort panel.

By Name	Sorts items alphabetically by name. This is the default setting.
By Package	Sorts items alphabetically by package name.
Unsorted	Displays items in the order they are defined in.
-	

For more information on sorting items, see Section 3.9.1 on page 35.

7.2 Examining slot information

When the class browser is first invoked, the default view is the slots view. You can also click the Slots tab to swap to it from another view. The slots view is shown in Figure 7.5.



Figure 7.5 Examining slots in the class browser

Section 7.1 on page 60 introduced you to the slots view in the class browser. This section gives a complete description of this view. For completeness, some information may be repeated. The areas available in the slots view are described below.

7.2.1 Class box

You enter the name of the class you want to browse in the Class text box. You can type in a class name explicitly, or you can transfer a class to the class browser using the **Class** standard action command in another tool, or by pasting a class in explicitly.

Note: You can use **Edit > Paste** to paste a class name into this area, even if the clipboard currently contains the string representation of the class name, rather than a class object itself. This lets you copy class names from other applications directly into the class browser. See Section 3.4.2 on page 25 for a complete description of the way the Common LispWorks clipboard operates.

7.2.2 Filter box

The Filter box lets you restrict the information displayed in the Slots list. See Section 3.12 on page 38 for a description of how to use the Filter box in any tool, and Section 7.1.3 on page 63 for an example of how to use it in the class browser.

7.2.3 Slots list

The largest section of the slots view lists the slot names of the current class. Selecting a slot in this list displays a description of it in the Description list, and you can operate on any number of selected slots using the commands in the **Works > Slots** menu.

If **Include Inherited Slots** is selected, slots inherited from the superclasses of the current class are listed as well as those explicitly defined on the current class. Deselect this button to see only those slots defined on the current class. In the slots/Functions tab of the Class Browser Preferences dialog, you can also configure the default setting of this option. Choose **Tools > Preferences** in the class browser to display this dialog.

The number of items listed in the Slots area is printed in the Matches box.

7.2.4 Description list

This list displays a description of the selected slot. The following information is printed:

From Classes	The classes that this slot is defined in.
Slot Name	The name of the slot.
Туре	The slot type.
Initargs	The initargs, if any, which can be used to refer to the slot.
Initform	The initform, or initial value, of the slot.
Readers	The readers of the slot. These are the names of any func- tions which can be used to read the current value of the slot.
Writers	The writers of the slot. These are the setf methods which may be used to change the slot value.
Allocation	The allocation of the slot.

To operate on any of the items displayed in this area, select them and choose a command from the **Works > Description** menu. This submenu contains the standard action commands described in Section 3.8 on page 31. You can operate on more than one item at once by making multiple selections in this area.

7.2.5 Performing operations on the current class

You can operate on the current class using the commands in the **Works** > **Classes** menu. The standard action commands described in Section 3.8 on page 31 are available in this submenu.

Choose **Works > Classes > Browse Metaclass** to select, and describe in the normal way, the class of the current class.

7.3 Examining superclasses and subclasses

The hierarchy view of the class browser lists the immediate superclasses and subclasses of the current class. This view can be useful for navigating the class hierarchy if you want to be able to see both superclasses and subclasses at the same time.

Click on the Hierarchy tab to browse classes with the hierarchy view. The hierarchy view shown in Figure 7.6 appears.

🕰 Class Browser 1	
Class: CAPI:CHOICE	✓ × 3
Hierarchy Subclasses Superclasses S	lots Initargs Functions Precedence
Superclasses:	Subclasses:
CAPI:COLLECTION	CAPI:BUTTON-PANEL
	CAPI:DOUBLE-LIST-PANEL
	CAPI: GRAPH-PANE
	CAPI:LIST-PANEL
	CAPI:MENU-COMPONENT
	CAPI: OPTION-PANE
	CAPI: TAB-LAYOUT
	CAPI::TEXT-INPUT-CHOICE
	CAPI::TOOLBAR-COMPONENT
	CAPI::TREE-VIEW
Description:	
Package: CAPI	
Name: PUSH-BUTTON-PAN	EL
Metaclass: STANDARD-CLASS	
Accessibility: :EXTERNAL	

Figure 7.6 Viewing superclass and subclass information in the class browser

The areas available in the hierarchy view are described below.

7.3.1 Class box

As with other views in the class browser, the name of the class being browsed is given here. See Section 7.2.1 on page 69 for more details.

7.3.2 Superclasses list

This list displays the superclasses of the current class. Double-clicking on any class makes it the current class.

Selecting a class in this list displays its description in the Description list.

7.3.3 Subclasses list

This list displays the subclasses of the current class. Double-clicking on any class makes it the current class.

Selecting a class in this list displays its description in the Description list.

7.3.4 Description list

This list displays a description of the first class selected in either the Superclasses or Subclasses lists, or the current class if there is no selection in either of these lists. The following information is printed:

Package	The name of the package that the selected class is defined in.
Name	The name of the selected class.
Metaclass	The metaclass of the selected class. The metaclass is the class of Lisp object which the current class belongs to.
Accessibility	The accessibility of the selected class—whether the symbol is external or internal, as returned by find-symbol.

To operate on any of the items displayed in this area, select them and choose a command from the **Works > Description** menu. This menu contains the standard actions commands described in Section 3.8 on page 31. You can operate on more than one item at once by making a multiple selection in this area.

7.3.5 Performing operations on the selected classes or the current class

You can use the **Works > Classes** menu to perform operations on any number of items selected in either the Subclasses area or the Superclasses area. If no items are selected, then the current class is operated on by the commands in this submenu. The standard actions commands described in Section 3.8 on page 31 are available in this submenu.

Choose **Works > Classes > Browse Metaclass** to select, and describe in the normal way, the class of the selected classes, or the current class.

Note: If more than one item is selected, and the command chosen from the **Works > Classes** menu invokes a tool which can only display one item at a time, then the extra items are added to the **History > Items** submenu of the tool, so that you can easily display them.

7.4 Examining classes graphically

As already mentioned, you can view class relationships graphically using either the superclasses or subclasses views. This gives an immediate impression of the class hierarchy, but contains no details about information such as slots, readers and writers.

Click on the Subclasses tab to browse subclasses in a graph, and click on the Superclasses tab to view superclasses in a graph. Except for the type of information shown, these two views are visually identical. The subclasses view is shown in Figure 7.7.



Figure 7.7 Viewing subclasses graphically in the class browser

The areas available in the subclasses and superclasses views are described below.

7.4.1 Class box

As with other views in the class browser, the name of the class being browsed is given here. See Section 7.2.1 on page 69 for details.

7.4.2 Subclasses and superclasses graphs

The main area of these views is a graph showing either the subclasses or the superclasses of the current class, depending on the view you have chosen. The generic facilities available to all graph views throughout the environment are available here: see Chapter 5, "Manipulating Graphs" for details.

Selecting a node in this displays a description of the class it represents in the Description list.

7.4.3 Description list

This list displays a description of the first class selected in the graph. This gives the same information as the Description list in the hierarchy and precedence views. See Section 7.3.4 for details.

7.4.4 Performing operations on the selected classes or the current class

You can operate on the selected node in the graph using the commands in the **Works > Classes** menu. If no node is selected, then the current class is operated on by the commands in this menu. The standard actions commands described in Section 3.8 on page 31 are available in this menu.

Choose **Works > Classes > Browse Metaclass** to select, and describe in the normal way, the class of the selected classes, or the current class.

7.4.5 An example

1. Examine the class capi:choice by typing capi:choice into the Class area of the class browser and pressing Return.

The class is described in the current view.

2. Click on the Subclasses tab in the class browser.

The relationships between capi:choice and its subclasses are shown in a graph, as in Figure 7.8.



Figure 7.8 Relationship between capi:choice class and its subclasses

By default, the subclasses of the current class are shown in the graph. To expand any node in the graph, double-click on it.

3. Double-click on the CAPI: BUTTON-PANEL node to see the subclasses of this class.

The classes of button panel object available are displayed in the graph, including the push button panel class that you saw in the examples in Section 7.1 on page 60.

4. To graph the superclasses, click the Superclasses tab.

The relationships between capi:choice and its superclasses are shown in a graph, as in Figure 7.9.

CAPI:COLLECTION — CAPI:CHOICE

Figure 7.9 Relationship between capi:choice class and its superclasses

7.5 Examining generic functions and methods

Click the Functions tab to examine information about the generic functions and methods defined on the current class. The functions view shown in Figure 7.10 appears.

🛃 Class Browser 1
Class: CAPI:CHOICE
Hierarchy Subclasses Superclasses Slots Initargs Functions Precedence
Methods 🔽 🗹 Include Inherited Methods/Functions
Filter: Katches: 160
WETHOD /SETE CADI.CALLEACUS_ACTION_CALLEACUA, /T CADI.CALLEA
(METHOD (SETE CADI-CALLBACKS-ACTION-CALLBACK, II CAPI-CALLBACK)
(METHOD (SETF CAPI: CALLBACKS EXTEND-CALLBACK) (T CAPI: CALLBA
(METHOD (SETF CAPI: CALLBACKS-RETRACT-CALLBACK) (T CAPI: CALLB,
(METHOD (SETF CAPI: CALLBACKS-SELECTION-CALLBACK) (T CAPI: CAL
(METHOD (SETF CAPI::CAPI-OBJECT-CLASS-PROPERTY) (T CAPI:CAPI
(METHOD (SETF CAPI:CAPI-OBJECT-NAME) (T CAPI:CAPI-OBJECT))
(METHOD (SETF CAPI-INTERNALS:CAPI-OBJECT-PLIST) (T CAPI:CAPI
(METHOD (SETF CAPI::CHOICE-KEEP-SELECTION-P) (T CAPI:CHOICE)
(METHOD (SETF CAPI:CHOICE-SELECTED-ITEM) (T CAPI:CHOICE))
(METHOD (SETF CAPI:CHOICE-SELECTED-ITEMS) (T CAPI:CHOICE))
(METHOD (SETF CAPI:CHOICE-SELECTION) (T CAPI:CHOICE))
(METHOD (SETF CAPI:CHOICE-SELECTION) :AROUND (T CAPI:CHOICE)
(METHOD (SETF CAPI::COLLECTION-DATA-FUNCTION) (T CAPI:COLLEC
(METHOD (SETF CAPI:COLLECTION-ITEMS) (T CAPI:COLLECTION))
(METHOD (SETF CAPI:COLLECTION-ITEMS) :AROUND (T CAPI:COLLECT.▼
Description:
Name: (METHOD (SETF CAPI:CALLBACKS-ACTION-CALLBACK) (T
Function: # <standard-writer-method (setf="" capi:callbacks-ac<="" td=""></standard-writer-method>
Lambda List: (VALUE CALLBACKS)
Documentation:
Source Files:

Figure 7.10 Displaying function information in the class browser

This view can be especially useful when used in conjunction with the generic function browser. The areas available are described below.

7.5.1 Class box

As with other views in the class browser, the name of the class being browsed is given here. See Section 7.2.1 on page 69 for more details.

7.5.2 Filter box

The Filter box lets you restrict the information displayed in the list of functions or methods. See Section 3.12 on page 38 for a description of how to use the Filter box in any tool, and Section 7.1.3 on page 63 for an example of how to use it in the class browser.

7.5.3 List of functions or methods

This lists each generic function or method defined on the current class. Items selected in this list can be operated on via the **Works > Methods** menu, as described in Section 7.5.6 on page 80. Double-clicking on a function or method displays its source code definition in the editor, if possible.

Select Methods or Generic Functions from the drop-down list box to choose which type of information to list. If **Include Inherited Methods/Functions** is checked, generic functions or methods inherited from the superclasses of the current class are also displayed. In the slots/Functions tab of the Class Browser Preferences dialog, you can also configure the default settings of these options in the class browser. Choose **Tools > Preferences** to display this dialog.

7.5.4 Description list

The list at the bottom of the tool gives a description of the function or method selected in the main list. The following information is shown:

Name	The name of the selected generic function or method.
Function	The function which the selected function or method relates to.

Lambda List	The lambda list of the selected generic function or method.
Documentation	The Common Lisp documentation for the selected func- tion or method, if any exists.
Source Files	The source files for the selected generic function or method.

To operate on any of the items displayed in this area, select them and choose a command from the **Works > Description** menu. This submenu contains the standard actions commands described in Section 3.8 on page 31. You can operate on more than one item at once by making a multiple selection in this area.

7.5.5 Performing operations on the current class

You can operate on the current class using the commands in the **Works** > **Classes** menu. The standard action commands described in Section 3.8 on page 31 are available from this submenu.

Choose **Works > Classes > Browse Metaclass** to select and describe the class of the current class.

7.5.6 Operations specific to the current function or method

In addition to the commands described above, the following commands are available when using the functions view.

The standard action commands described in Section 3.8 on page 31 are available from the **Works > Methods** menu.

Choose **Works > Methods > Undefine** to remove the selected functions or methods from the environment. You are prompted before the functions or methods are removed.

Choose **Works > Methods > Trace** to display the **Trace** submenu available from several tools. This submenu lets you trace the selected methods or generic functions. A full description of the commands in this submenu is given in Section 3.10 on page 36.

7.6 Examining initargs

Click the Initargs tabs to examine information about the initargs of the current class. The initargs view shown in Figure 7.11 appears.

🛃 Class Browser 1	- 🗆 ×
Class: CAPI:CHOICE	✓ × 3
Hierarchy Subclasses Sup	erclasses Slots Initargs Functions Precedence
Filter:	✓ × Matches: 25
ACCEPTS-FUCUS-P	·
:CALLBACK-TYPE	
:DATA-FUNCTION	
:DO-CACHE	
:EXTEND-CALLBACK	
:INITIAL-FOCUS-ITEM	
: INTERACTION	
TTEMS	
TTEMS_COUNT_FUNCTION	r
• ITEMS-GET-FUNCTION	-
Description:	
Initarg: :	ACTION-CALLBACK
Default Initarg:	
Default From Class:	
From Classes: C	API:CALLBACKS
Slot Name: C	API::ACTION-CALLBACK
Tritarge: 1	ACTION_CALLEACE
Tnitform: N	ITI.
Readers: C	API:CALLBACKS-ACTION-CALLBACK
Writers: (SETF CAPI:CALLBACKS-ACTION-CALLBACK)
Allocation: :	INSTANCE
Π	

Figure 7.11 Displaying initarg information in the class browser

The initargs of a class are similar to the slots, except that initargs give you a more precise representation of the values that may be assigned to an instance of a class, for instance using the writers for that class.

The areas available are described below.

7.6.1 Class box

This area gives the name of the class being browsed. See Section 7.2.1 on page 69 for details.

7.6.2 Filter box

The Filter box lets you restrict the information displayed in the initargs list. See Section 3.12 on page 38 for a description of how to use the Filter box in any tool, and Section 7.1.3 on page 63 for an example of how to use it in the class browser.

7.6.3 List of initargs

This lists the slots in the current class for which initargs have been defined. Selecting an item in this list displays information in the Description list. Any items selected can also be operated on via the **Works > Slots** menu.

7.6.4 Description list

This area gives a description of the initarg selected in the Initargs area. The following items of information are displayed:

Initarg	The name of the selected initarg.
Default Initarg	The default value for the selected initarg, if defined with :default-initargs.
Default From C	lass
	The class providing the default for the initarg.
From Classes	The class from which the selected initarg is inherited

Slot Name	The name of the slot to which this initarg relates.
Туре	The type of the selected initarg.
Initargs	All initargs applicable to the same slot
Initform	The initform for the slot which is represented by the selected initarg.
Readers	The readers for the slot which is represented by the selected initarg.
Writers	The writers for the slot which is represented by the selected initarg.
Allocation	The allocation for the selected initarg. See CLOS in the ANSI Common Lisp specification.

Items selected in this list can be operated on via the Works > Description menu.

7.6.5 Performing operations on the current class

You can operate on the current class using commands in the **Works > Classes** menu. The standard action commands described in Section 3.8 on page 31 are available in this submenu.

Choose **Works > Classes > Browse Metaclass** to select, and describe in the normal way, the class of the current class.

7.7 Examining class precedences

Click the Precedence tag to examine information about the precedence list of the current class. The precedence list is used to generate the method combinations for a class, and thus can be used to tell you which method applies in a given case. The precedence view shown in Figure 7.12 appears.

🛃 Class Browser 1
Class: CAPI:CHOICE
Hierarchy Subclasses Superclasses Slots Initargs Functions Precedence
Filter: Matches: 8
CAPI: CHOICE
CAPI: COLLECTION
CAPI::SIMPLE-ELEMENT
CAPI::BASIC-ELEMENT
CAPI: CALLBACKS
CAPI: CAPI-OBJECT
SIANDARD-UBUELI
1
Description:
Package: CAPI
Name: CHOICE
Metaclass: STANDARD-CLASS
Accessibility: :EXTERNAL

Figure 7.12 Displaying precedence information in the class browser

The areas available are described below.

7.7.1 Class box

As with all other views in the class browser, the current class is printed in this area. See Section 7.2.1 on page 69 for full details of its use.

7.7.2 Filter box

The Filter box lets you restrict the information displayed in the list of precedences. See Section 3.12 on page 38 for a description of how to use the Filter box in any tool, and Section 7.1.3 on page 63 for an example of how to use it in the class browser.

7.7.3 List of precedences

This list is the class precedence list of the current class. Precedences are listed highest first. Double-clicking on an item in this list describes that class in the class browser.

7.7.4 Description list

This gives the same class description available in the superclasses, subclasses, and hierarchy views. See Section 7.3.4 on page 73 for details.

7.7.5 Performing operations on the selected classes or the current class

You can operate on any number of selected items in the list of precedences using the commands in the **Works > Classes** menu. If no items are selected, then the current class is operated on by the commands in this submenu. The standard actions commands described in Section 3.8 on page 31 are available in this submenu.

Choose **Works > Classes > Browse Metaclass** to select, and describe in the normal way, the class of the selected classes, or the current class.

Note: If more than one item is selected, and the command chosen from the **Works > Classes** menu invokes a tool which can only display one item at a time, then the extra items are added to the **History > Items** submenu of the tool, so that you can easily display them.

The Class Browser

The Compilation Conditions Browser

8.1 Introduction

The compilation conditions browser gives you an interface to the warning and error conditions you are likely to encounter when compiling your source code. It allows you to see the relationship between different errors or warnings encountered during compilation, and gives you immediate access to the source code which produced those errors or warnings.

You can use it to view the conditions signaled during compilation of files from any part of the environment: whether you are compiling files using the system browseror indeed the editor, any ensuing errors can be displayed in the compilation conditions browser.

The compilation conditions browser has two views.

- The text view, which gives you a list of the errors, grouping together errors of a similar type, and from the same file.
- The output view, which can be used to display the output messages in the environment.

To create a compilation conditions browser, choose **Tools > Compilation Conditions Browser** from the LispWorks podium.

8.2 Examining error conditions

The text view is the default view in the compilation conditions browser, and is therefore visible when the browser is first invoked. The compilation conditions browser appears as shown in Figure 8.1.

🛃 Compilation Condition	ns Browser 2		_ 🗆 ×
Text Output			
<u>F</u> ilter:		🖌 🗙 Match	nes: 2
Message		Name	File
PALN::*LAYOUT-ALGORIT V is bound but not reference	"HM* assumed special ed	(TOP-LEVEL-FORM 2) PALN::DO-NUMBERS-	foo.lis FR foo.lis
Description:			
Ready.			

Figure 8.1 The compilation conditions browser

The text view has a filter area and two description areas, as described below.

8.2.1 Filter area

This area lets you restrict the information displayed in the compilation conditions area so that only the conditions you are interested in are shown. See Section 3.12 on page 38 for details about how to use the Filter area in a tool.

8.2.2 Compilation conditions area

The compilation conditions area displays a list divided into three columns.

• The first column shows the condition itself.

- The second column names the form in which the condition was signaled.
- The third column shows the filename in which the condition was signaled.

Note that you may have to scroll the compilation conditions area horizontally in order to see all three columns.

Double-click on any item to display the form that signaled the condition in the editor.

8.2.3 Description area

The description area shows a description of any item selected in the compilation conditions area containing more detailed information about the conditions selected in the compilation conditions area. The following information is listed:

Condition	The error condition for the selected item in the message area.
Class	The class of the selected condition.
Definition	The name of the form in which the condition was sig- naled.
File	The name of the file that contains the Lisp source code that caused the selected condition.

Items selected in this area may be examined using the **Works > Description** menu which allows a variety of LispWorks tools to be invoked on the selected item in the description area.

8.3 Examining output

Click on the output tag to switch to the output view of the compilation conditions browser. The output view is similar to that seen in an output browser. See Chapter 17, "The Output Browser" for more information.

8.4 Configuring the display

The manner in which the compilation conditions browser displays information can be customized by selecting **Tools > Preferences** in the browser, to invoke the compilation conditions browser preference dialog. When first invoked, it appears as shown in Figure 8.2, with the general view active.

L Compilation Conditions Browser 2 Preferences	×
General Display	
Package	
COMMON-LISP-USER	🖌 🔭 📗
Show Package Names	
<u> </u>	Apply

Figure 8.2 The compilation conditions browser preferences dialog

8.4.1 Package names preferences

In the general view select **Show Package Names** to toggle display of packages in all references to symbols. Use the package selection box to select which package to show. The default package is **CL-USER**. Setting the right package and turning off package names can simplify a complicated list to a large degree.

8.4.2 Pathname preferences

In the display view, the preferences dialog appears as in Figure 8.3.

L Compilation Conditions Browser 2 Prefer	ences 🗙
General Display Pathnames Show Full Pathname Show Leaf Pathname Conditions : Types to Display	
<u> </u>	ncel <u>Apply</u>

Figure 8.3 The display view of the preferences dialog

The major part of this dialog is taken up by the pathnames selection area, which has two radio buttons

Check **Show Full Pathname** to show the full pathname of all files displayed. This is the default setting.

Check **Show Leaf Pathname** to show just the filename and relative pathname of all files displayed, and omit the full pathname.

8.4.3 Condition type preferences

Clicking on **Types to Display...** in the display view of the preferences dialog calls up the condition types dialog. It consists of a standard filter box and two lists.

• The Selected Types list shows the error types that are displayed in the browser.

• The Unselected Types list shows the error types that are not displayed in the browser.

By default, all condition types are displayed.

To remove an error type from the Selected Types list, select it in the Selected Types list and click on <<<. It is transferred into the Unselected Types list.

To add an error type back into the Selected Types list, either:

- select it in the Unselected Types list and click on >>>, or
- type its name in the Select Type area and press Return.

If you use the second of these methods, note that you can press the Tab key at any point to either complete the name of the error type, or display a dialog listing all the possible completions, as appropriate. If a dialog appears, double-click on any item to select it.

To display all error types, click on AII.

To display no error types, click on **None**. If you only want to display a few error types, click on this button and then transfer the error types you do want to see into the Selected Types list using one of the methods described above.

When you have finished choosing the error types, click on **OK**. The dialog is dismissed, and the compilation conditions browser is updated to display the error types you have requested.

Click on **Cancel** to cancel the dialog. The dialog is dismissed, and no changes are made to the display.

8.5 Access to other tools

Like many other tools in the environment, items selected in the compilation conditions browser may be transferred into other tools for further examination.

Items selected in the Description area may be examined using the **Works > Description** menu. See Section 3.8.1 on page 32 for more information on the operations available from this menu.

The Debugger Tool

When developing source code, mistakes may prevent your programs from working properly, or even at all. Sometimes you can see what is causing a bug in a program immediately, and correcting it is trivial. For example, you might make a spelling mistake while typing, which you may instantly notice and correct.

More often, however, you need to spend time studying the program and the errors it caused before you can debug it. This is especially likely when you are developing large or complex programs.

A debugger tool is provided to make this process easier. This tool is a graphical front-end to the command line debugger which is supplied with your Lisp image. In order to get the best use from the debugger tool, it is helpful if you are familiar with the command line debugger supplied.

The debugger tool can be used to inspect the behavior of programs which behave in unexpected ways, or which contain Common Lisp forms which are syntactically incorrect. The command line debugger is invoked automatically whenever errors occur. There are two ways that you can invoke the debugger tool:

- If you have entered the command line debugger by evaluating code that contains bugs in a listener, choose Debug > Listener > Start GUI Debugger to invoke the debugger tool.
- If code containing bugs is run from another source (for example, as a result of running a windowed application, or compiling code in a file of source code), a notifier window appears. Click on the **Debug** button in this notifier to invoke the debugger tool.

The command line debugger can be entered by signaling an error in interpretation or execution of a Common Lisp form. For each error signaled, a further level of the debugger is entered. Thus, if, while in the debugger, you execute code which signals an error, a lower level of the debugger is entered. The number in the debugger prompt is incremented to reflect this.

For example:

1. Define the following function in the listener.

```
(defun thing (number)
(/ number 0))
```

This function which attempts to divide a number given as an argument by zero.

2. Now call this function as follows:

(thing 12)

The call to thing invokes the command line debugger.

- **3.** Choose **Debug > Listener > Start GUI Debugger** to invoke the debugger tool.
- **4.** For now, click **Abort** at the bottom of the debugger tool to return to the top level event loop in the listener.

Note that you can also invoke the command line debugger by tracing a function and forcing a break on entry to or exit from that function. See Section 3.10 on page 36 for details.

9.1 Description of the debugger

The debugger tool has only one view, and is shown in Figure 9.1 below.

or condition.					Debugger I	backtrace.
\					/	/
					/	
🗶 Debugge	er 1				,	
<u>C</u> ondition	<u>F</u> rame	<u>V</u> ariables	<u>R</u> estarts	\$		
Condition:						
We've reache	d fifty			/	/	E
Backtrace:						
TWYOKE-DEB	HCCED			_/		
BRFAK	OGGER					
TEST			/	/		
("Subprimi	tive «S	nearest	~S." SV9	STEM···	EVAL 349	1
SVSTEM··DO	-FVALUZ	TTON	~»»• »I×		SEVAD DAD	/
SVSTEM ·· ST	RURIER RURI-GO'	TL - TNTEDNA	T.			
GYGTEN ST	OF-LEVE OD IEUE	56-101660A	. ш			
SISIEM::SI	OF-LEVE CTENED	SD TOD IEIEI				
JULICE COLUMN	DIENER-	-IOF-LEVEL	TOD IFIT			
LISPWORKS-	10072::	LISIENER-	TOP-LEVE	ST-LONI	JIION	
CAPI::INIT	IALIZE-	-AND-PRUCE	SS-EVENI	rs		
(SUBFUNCTI	ON 1 CA	API::CALL-	AND-PROC	CESS-EV	VENTS)	
(SUBFUNCTI	ON 1 MH	P::INITIAL	IZE-PROC	CESS-ST	ГАСК)	
SYSTEM::%%	FIRST-C	CALL-TO-ST	ACK			
Variables:						
LOOP::IT	Т					
#: by-23	1					
#: to-22	100					
I	50					
TOTAL /	1275					
	_	- 1		1		
	Report	Bug I	Continue	1	Abort /	
to of variables	for					
e of variables		Echo area.		Cont	trol buttons	
ucu name.						

Figure 9.1 Debugger tool

The debugger tool has three areas, and a row of buttons at the bottom. These are described below. If you invoke the debugger tool by clicking **Debug** in a notifier window, the tool also contains a listener pane. This provides you with a useful way of evaluating Common Lisp forms interactively in the context of the bug.

9.1.1 Condition box

This area displays the error condition which caused entry to the debugger. You cannot edit the text in this box.

The error condition can be operated on by commands in the **Debug > Condition** menu. See Section 9.6 on page 101 for details.

9.1.2 Backtrace list

The backtrace list displays the function calls on the debugger stack. Each item, referred to as a frame on the stack, shows a separate function call.

Selecting any frame displays any variables associated with that function in the Variables area.

Double-clicking on any frame finds and displays the source code definition for that function in the editor, prompting you for a tags file first, if necessary.

Any item selected in this area can be operated on using the commands in the **Debug > Frame** menu: See Section 9.7 on page 102 for details.

9.1.3 Variables list

This list displays any variables associated with the function selected in the Frame area, together with their values for that frame in the stack.

Double-click on any variable to inspect it.

Any items selected in this area can be operated on using the commands in the **Debug > Variables** menu: see Section 9.8 on page 102 for details.

9.1.4 Buttons

At the bottom of the debugger tool is a row of buttons, as described below.
Click **Report Bug** if you wish to report a bug to Harlequin via electronic mail. Clicking on this button prompts you for a short description of the bug, and then invokes an editor into which you can type your bug report. See Section 29.1 on page 376 for more details.

Click **Continue** to return from the debugger and invoke the continue restart.

Click Abort to return from the debugger and invoke the abort restart.

9.2 What the debugger tool does

The debugger tool provides a number of important facilities for inspecting programs.

Common Lisp, like most programming languages, uses a stack to store data about programs during execution. The debugger tool allows you to inspect and change this stack to help get your programs working properly.

You can use it to trace backwards through the history of function calls on the stack, to see if the program behaves as expected, and locate points at which things have gone wrong.

You can also inspect variables within those functions, again to verify that the program is doing what is expected of it.

The debugger tool also allows you to change variables on the stack. This is useful when testing possible solutions to the problems caused by a bug. You can run a bugged program, and then test fixes within the debugger tool by altering values of variables, and then resuming execution of the program.

9.3 Simple use of the debugger tool

When you enter the debugger tool, the Condition area displays a message describing the error. The **Debug > Restarts** menu lists a number of *restart* options, which offer you different ways to continue execution.

- 1. For example, type the name of a variable which you know is unbound (say fubar) at the listener prompt.
- 2. Choose Debug > Listener > Start GUI debugger to enter the debugger tool.
- 3. Select the **Debug > Restarts** menu to display the options available.

A number of restarts are displayed that offer you different ways in which to proceed. These are the same options as those displayed at the command line debugger before you invoked the debugger tool.

Two special restarts can be chosen: the *abort* and *continue* restarts. These are indicated by the prefixes (abort) and (continue) respectively. As a shortcut, you can click on the Abort or Continue buttons to invoke them, instead of choosing the appropriate menu command.

In the case of the continue restart, different operations are performed in different circumstances. In this example, you can evaluate the form again. If you first set the variable to some value, and then invoke the continue restart, the debugger is exited.

4. In the listener, set the value of fubar as follows:

(setq fubar 12)

5. Click Continue in the debugger tool.

The debugger tool disappears, and the command line debugger is exited in the listener, and the value 12 is returned; the correct result if the variable had been bound in the first place.

You can also click **Abort** to invoke the abort restart. This restart always exits the current level of the debugger and returns to the previous one, ignoring the error which caused the present invocation of the debugger.

In general, you should use the continue restart if you have fixed the problem and want to continue execution, and the abort restart if you want to ignore the problem completely and stop execution.

9.4 The stack in the debugger

As already mentioned, the debugger tool allows you to examine the state of the *execution stack*, which is listed in the Backtrace area. This area consists of a sequence of *stack frames*. A stack frame is a description of some part of a program, or something relating to the program, which is packaged into a block of memory and placed on the stack during program execution. These frames are not directly readable without the aid of the debugger. There can be frames on the stack representing active function invocations, special variable bindings, restarts, and system-related code. In particular, the execution stack has a *call frame* for each active function call. That is, it stores information describing calls of functions which have been entered but not yet exited. This includes information such as the arguments with which the functions were called. By default, only call frames for active function calls are displayed in the Backtrace area. See Section 9.9 on page 103 for details of how to display other types of call frame.

The top of the stack contains the most recently-created frames (and so the innermost calls), and the bottom of the stack contains the oldest frames (and so the outermost calls). You can examine a call frame to find the name of a function, and the names and values of its arguments, and local variables.

9.5 An example debugging session

To better understand how you can make use of the debugger, try working through the following example session. In this example, you define the factorial function, save the definition to a file on disk, compile that file and then call the function erroneously.

1. Choose File > New or click on \square .

A new file is created and displayed in the editor. If you have not already invoked the editor, it is started for you automatically.

2. In the new file, define the function fac to calculate factorial numbers.

```
(defun fac (n)
(if (= n 1) 1
(* n (fac (- n 1)))))
```

- **3.** Choose File > Save or click on , and type a filename when prompted in the echo area of the editor.
- **4.** Choose **File > Compile and Load** to compile the file and load it into the environment.

The editor switches to the output view while compilation takes place. When prompted, press Space to return to the text view. The fac function is now available to you for use in the environment.

5. In the listener, call fac erroneously with a string argument.

```
(fac "turtle")
```

The environment notices the error: The arguments of = should be numbers, and one of them is not.

6. Choose **Debug > Listener > Start GUI Debugger** to invoke the debugger tool.

Take a moment to examine the backtrace that is printed in the Backtrace area.

7. Starting from the selected frame, select the next three frames in the Backtrace area in turn to examine the state of the variables which were passed to the functions in each call frame. Pay particular attention to the fac function.

The error displayed in the Condition box informs you that the = function is called with two arguments: the integer 1 and the string "turtle". Clearly, one of the arguments was not the correct type for =, and this has caused entry into the debugger. However, the arguments were passed to = by fac, and so the real problem lies in the fac function.

In this case, the solution is to ensure that fac generates an appropriate error if it is given an argument which is not an integer.

8. Double-click on the line FAC in the Backtrace area of the debugger tool.

The editor appears, with the cursor placed at the beginning of the definition for fac. Double-clicking on a line in the Backtrace area is a shortcut for choosing **Debug > Frame > Find Source**.

9. Edit the definition of the fac function so that an extra if statement is placed around the main clause of the function. The definition of fac now reads as follows:

```
(defun fac (n)
 (if (integerp n)
    (if (= n 1) 1
        (* n (fac (- n 1))))
    (print "Error: argument must be an integer")))
```

The function now checks that the argument it has been passed is an integer, before proceeding to evaluate the factorial. If an integer has not been passed, an appropriate error message is generated.

- **10.** Choose File > Save and File > Compile and Load again, to save, recompile and load the new definition into the environment.
- **11.** Click on the **Abort** button in the debugger tool, to destroy the tool and return the listener to the top level loop.
- **12.** In the listener, type another call to fac, once again specifying a string as an argument. Note that the correct error message is generated.

This next part of the example shows you how you can use the various restarts which are listed as commands in the **Debug > Restarts** menu.

1. Call fac again with a new argument, but this time type the word length incorrectly.

(fac (legnth "turtle"))

Choose Debug > Listener > Start GUI Debugger to invoke the debugger tool.

You can spot immediately what has gone wrong here, so the simplest strategy is to return a value to use.

- **3.** Choose **Debug > Restarts > Return some values from the call to LEGNTH**. You are prompted for a form to be evaluated in the listener.
- 4. Type 6, which is the value that would have been returned from the correct call to (length "turtle").

Having returned the correct value from (length "turtle"), fac is called with the correct argument and returns the value 720.

9.6 Performing operations on the error condition

You can perform operations on the error condition that caused entry into the debugger using the commands available in the **Debug > Condition** menu.

The standard action commands are available in the **Debug > Condition** menu. For more details about these commands, see Section 3.8 on page 31.

Choose **Debug > Condition > Report Bug** to submit a bug to Harlequin by electronic mail.

9.7 Performing operations on frames in the stack

Any frame selected in the Backtrace list can be operated on using commands in the **Debug > Frame** menu. This menu is also available as a popup from the backtrace list itself. The commands available allow you to operate on the function displayed in each frame.

Choose **Debug > Frame > Find Source** to search for the source code definition of the object pointed to by the current frame. If it is found, the file is displayed in the editor: the cursor is placed at the start of the definition. See Chapter 10, "The Editor" for an introduction to the editor.

Choose **Debug > Frame > Documentation** to display the Common Lisp documentation for the object pointed to by the current frame, if any exists. It is printed in a special output browser window.

Choose **Debug > Frame > Restart Frame** to continue execution from the selected restart frame. The action that is taken when choosing this command is printed with each restart frame in the Backtrace area. Note that restart frames must be listed for this command to be available: see Section 9.9.1 for details.

Choose **Debug > Frame > Return From Frame** to resume execution from the selected frame. You are prompted for a value to return from the selected frame. This option allows you to continue execution smoothly after you have dealt with the error which caused entry into the debugger.

Choose **Debug > Frame > Trace** to display the standard Trace menu. This lets you trace the function in the selected frame in a variety of ways: see Section 3.10 on page 36 for details.

9.8 Performing operations on variables in a frame

There are two commands which you can choose to perform operates on any variables selected in the Variables area. These are both available in the **Debug > Variables** menu or from the popup menu of the variables list itself.

The standard action commands are available in the **Debug > Variables** menu. For more details about these commands, see Section 3.8 on page 31. Choose **Debug > Variables > Set** to set the values of any variables selected in the Variables area. A dialog prompts you to enter a new value to which the variables are set.

9.9 Configuring the debugger tool

You can control the behavior of the debugger using the Debugger Preferences dialog, which appears when you choose **Tools > Preferences** from the debugger.



Figure 9.2 Debugger Preferences dialog

9.9.1 Configuring the call frames displayed

By default, the call frame for each active function call in the backtrace is listed in the Backtrace area. There are a number of other types of call frame which are hidden by default. Displac call frames of these types by selecting them in the View Frame panel of the debugger Preferences dialog.

Bindings	Displays all the binding frames in the Backtrace list.				
Catchers	Lists the catch frames in the Backtrace list.				
Handlers	Lists the handler frames in the Backtrace list.				
Hidden SymbolsLists any hidden symbols in the Backtrace list.					

RestartsLists all the restart frames in the Backtrace list. Each
restart frame is listed, with the restart action to be taken
given in brackets. To restart execution at any restart
frame, select the frame, and choose Debug > Frame >
Restart Frame.Restart Frame.

Invisible Functions

Lists all invisible frames (such as the call to the error function itself) in the Backtrace list.

Note that all these commands can be toggled: choosing any command switches the display option on or off, depending on its current state. By default, all the options are off when the debugger is first invoked.

9.9.2 Configuring the symbols displayed

You can configure the debugger tool so that only certain symbols are displayed in the Backtrace area. For any package in the environment, you can choose to do any of the following:

- Hide all symbols in the package.
- Hide all internal symbols in the package.
- Display all symbols in the package.

To configure the display of symbols, choose **Change Hidden Packages** from the Packages panel of the Debugger Preferences dialog.

This dialog contains three lists, as follows:

- The Show All Symbols list shows the packages from which all symbols are displayed in the Backtrace area.
- The Show External Symbols Only list shows those packages from which internal symbols are be displayed.
- The Hide All Symbols list shows those packages from which no symbols are displayed in the Backtrace area.

To move packages from one list to any other, select them and click on >>> or <<< as appropriate.

When you have finished changing the configuration, click on **OK** to accept the changes, or **Cancel** to leave the settings as they are.

9.9.3 Displaying package information

As with other tools, you can configure the way package names are displayed in the debugger tool in the Package panel of the Debugger Preferences dialog.

Check **Show Package Names** to turn the display of package names in the Back-trace and Variables lists on and off.

Specify a package name in the text box to change the process package of the debugger tool. Click on \mathfrak{P} to complete the package name as far as possible. By default, the current package is the same as the package from which the error was generated.

The Debugger Tool

10

The Editor

The environment has a text editor which is designed specifically to make writing Lisp source code easier. It is very similar to the EMACS text editor, and you should refer to the *Editor User Guide* supplied with your software, for a full description of the extensive range of functions and commands available.

The editor features a comprehensive set of menus, as well as a number of different views, and it has an interface which is completely consistent with all the other tools in the environment. This chapter gives a complete description of these aspects of the editor, as well as giving you a general overview of how the editor is used. If you have not used EMACS before, this chapter tells you all you need to know to get started.

The advantage of the editor is its ability to perform a wide range of operations by using menu commands, as well as the keyboard commands described in more detail in the *Editor User Guide*. These operations range from simple tasks such as navigating around a file, to more complex actions which have been specifically designed to ease the task of writing Lisp code.

By becoming familiar with the menu commands available, you can learn to use the editor effectively in a very short space of time, before moving on to more advanced operations. Like other tools, the editor offers a number of different views, which you can switch between using the tabs at the top of the editor window. Unlike other tools, one view in particular is used more often than any other.

- The text view is the most commonly used view in the editor. This lets you read and edit text files which are stored in your filesystem.
- The output view lets you examine any output messages from the environment.
- You can edit many different files at once in the same editor. The buffers view provides a quick way of navigating between different files that you have open.
- The definitions view is a convenient way of seeing the classes, functions, macros, variables and so on that are defined in the current file.
- Files may contain many definitions. The find definitions view lets you search for particular definitions of interest across many files.

You can create an editor using any of the following methods:

- Choose **Tools > Editor** from the LispWorks podium.
- Choose File > Open from the LispWorks podium, or click on in the button bar, and choose a filename in the dialog that appears.
- Make the listener the active window, and press Ctrl+x Ctrl+F. Type in the name of a file that you want to edit, including its drive and full pathname if it is not in the current directory.

Notice that when you create an editor from the **Tools** menu, you are not actually editing a file immediately.

10.1 Displaying and editing files

The text view is the default view in the editor, and is the one which you will become most familiar with. In this view, a buffer containing the text of the current file is displayed, and you can move around it and change its contents as you wish, then save it back to the original file (assuming that you have permission to write to it). The text view is automatically displayed when you first invoked the editor, and you can click on the Text tab to switch back to it from any other view. Figure 10.1 below shows an editor in the text view with a file open.

🛃 E dite	or 2 - fa	ct.lisp			
Text	Output	Buffers	Definitions	Changed Definition	s Find Definitions
(in-p (defu (if (ackage n fact (zerop * n (fa	"CL-US (n) o n) 1 act (1-	ER")		
LATIN-1	-**- fact.lis	p (CL·L	ISER) (Lisp) ()+100 c:\TEMP\fac	t.lisp

Figure 10.1 Text view in the editor

The text view has two areas, described below.

10.1.1 The editor window

The editor window is the main part of the editor. The text of the current file is shown in this area. As in many other editors, a block cursor is used to denote the current position in the file. Text is entered into the file at this position when you type.

To move the cursor to a particular point in the file, you can use any combination of the following methods:

- Position the cursor by moving the mouse pointer and selecting the point at which you want to place the cursor.
- If the file is too large to display all of it in the editor window, use the scroll bars to move up and down the file.
- Use any of the numerous keyboard commands that are available for navigating within a file.

If you are unfamiliar with the editor, you can use the first two methods to begin with. As you become more familiar, you will find it is often quicker to use the keyboard commands described in the *Editor User Guide*. Some of the most basic commands are also described in this chapter, in Section 10.8 on page 117.

10.1.2 The echo area

Underneath the editor window is an echo area, identical to the echo area in the other tools. This is used by the editor to display status messages, and to request more information from you when necessary. The echo area is contained in every view in the editor.

Whenever you invoke a command which requires further input (for instance, if you search a file for a piece of text, in which case you need to specify the text you want to search for), you are prompted for that input in the echo area. Type any information that is needed by the editor, and the characters you type are echoed in the echo area.

For many commands, you can save time by using *tab completion*. When you have partially specified input in the echo area, you can press the Tab key and the editor attempts to complete what you have typed. If it cannot complete the string uniquely, a dialog appears which lists all the possible alternatives. Double-click on any item in this dialog to place it in the echo area.

For example, suppose you have three files in the current directory, test1.lisp, test2.lisp and test3.lisp, and you want to edit test2.lisp using keyboard commands. Type ctrl+x ctrl+F, then type test and press Tab. A list appears which shows all three files. To edit test2.lisp, doubleclick on the item marked test2.lisp in this list. To get the hang of when tab completion is appropriate and when it is not, experiment by pressing the Tab key when specifying input in the echo area. As a rule, if there are a finite number of things you could type, then tab completion is appropriate. Thus, when opening a file already on disk, tab completion is appropriate (there is a finite number of files in the current directory). When specifying a string to search for, however, tab completion is not appropriate (you could specify practically any string).

10.1.3 Using keyboard commands

A full description of the keyboard commands available in the editor is beyond the scope of this manual, and you are advised to study the *Editor User Guide* to gain a full appreciation of the capabilities of the editor. However, of necessity, certain basic keyboard commands are discussed in this chapter. See Section 10.8 on page 117 of this manual for a brief introduction to some of the most important ones. The menu commands available are described throughout the rest of this chapter.

As with other keyboard commands used in the environment, the keyboard commands used in the editor are invoked by using a combination of the Control, Shift and Escape or Alt keys in conjunction with the letters of the alphabet. Some of the commands available perform the same, or a similar task as a menu command.

Each keyboard command in the editor is actually a shortcut for an *extended editor command*. You can invoke any extended command by typing its command name in full, preceded by the keyboard command Alt+x. Thus, to invoke the extended command **Visit Tags File**, type Alt+x visit tags file followed by Return. Case is not significant in these commands, and tab completion (described in Section 10.1.2 on page 110) may be used to avoid the need to type long command names out in full. This method is often useful if you are not certain what the keyboard shortcut is, and there are many extended commands which do not have keyboard shortcuts at all.

Many of the keyboard commands described in this chapter and in the *Editor User Guide* also work in the listener. Feel free to experiment in the listener with any of the keyboard commands that are described.

10.2 Displaying and swapping between buffers

The contents of the editor window is usually referred to as the *buffer*. Technically speaking, when you edit a file, its contents are copied into a buffer which is then displayed in the window. You actually edit the contents of the buffer, and never the file. When you save the buffer, its contents are copied back to the actual file on disk. Working in this way ensures that there is always a copy of the file on disk—if you make a mistake, or if your computer crashes, the last saved version of the file is always on disk, ensuring that you do not lose it completely.

Because of this distinction, the term buffer is used throughout, when referring to the text in the window.

An editor can only have one editor window, although there can be many buffers open at once. This means that you can edit more than one file at once, although only one buffer can be displayed at a time in the window—any others remain hidden.

The diagram below shows the distinctions between the window, buffers and files on disk.



Figure 10.2 Distinctions between the window, buffers, and files on disk

The buffers view allows you to display a list of all the buffers that are currently open in the editor, and gives you an easy way of navigating between them. Click on the Buffers tab to switch to this view, or press Ctrl+x Ctrl+B. The editor appears as shown in Figure 10.3 below.

🛃 Editor 2 - fact.lisp								
Text	Output B	Output Buffers		s 🛛 Chang	Changed Definitions		Find Definitions	
Filter:						< X	Matches:	1
A.,	Name	M	ode	Size	Pat	hname		
 %% 	fact.lisp foo.lisp lww-configure.li Main	Lis Lis sp Lis Fui	p p ndamental	83 5507 14092 0	c:\1 d:\u z:\c NIL	EMP\fact isers\nick\ lc\config\l	lisp .foo.lisp ww-configu	re.lisp
	-1 -××- fact lisp	{CL-11	ISEB} (Lise	<u>ა) በ+100 (</u>		MP\fact li		Þ
	1 Jocusp	,020		,,,		ini idoci	401	

Figure 10.3 Listing buffers in the editor

The buffers view has two areas, described below.

10.2.1 Filter box

You can use this area to restrict the number of buffers displayed in the Buffers area. See Section 3.12 on page 38 for details about how to use the Filter box in a tool.

10.2.2 Buffers area

Double-click on any buffer to display it in the editor using the text view. Buffers selected in this list can be operated on by commands in the **Works > Buffers** menu. See Section 10.13 on page 129 for more details.

10.3 Displaying Common Lisp definitions

The definitions view lists all the Common Lisp definitions which can be found in the current buffer. Click on the Definitions tab to switch to this view. The editor appears as shown in Figure 10.4 below.



Figure 10.4 Examining Common Lisp definitions in the editor

The definitions view has two areas, described below.

10.3.1 Filter box

You can use this area to restrict the number of definitions displayed in the definitions area. See Section 3.12 on page 38 for details about how to use the Filter box in a tool.

10.3.2 Definitions area

Double-click on any definition in this area to display its source code in the editor window. Definitions selected in this area can be operated on using commands in the editor's **Works > Definitions** menu. See Section 10.13.6 on page 133 for complete details of the commands available.

10.4 Finding definitions

Use the Find Definitions view to locate definitions with a given name. Type the name of the definition you are searching for in the name area and press Return or click on to display the matches and their locations. Double-click on a match to display the source. In addition, after using the Find Source editor command (bound to Alt+.), the Find Definitions view can be invoked using Alt+x View Source Search to give a complete list of the matches.

10.5 Changed definitions

The Changed Definitions view is identical to the Definitions view, except that only definitions that have been edited in the current session are listed. Click on the Changed Definitions tab to see this view. See the *Guide to the Editor* for more information on Buffer Changed Definitions and related commands.

10.6 Displaying output messages in the editor

As with several other tools, the editor provides an output view which can be used to examine any output messages which have been generated by the environment. Click on the Output tab to switch to this view. See Chapter 17, "The Output Browser", for more information about this view.

10.7 Setting editor preferences

You can configure the way in which items are listed in buffers, definitions and find definitions views using the Editor Preferences dialog. Choose **Tools > Preferences** within an editor to display this dialog.



Figure 10.5 Editor Preferences dialog

10.7.1 Sorting items in lists

By default, items in the buffers, definitions and find definitions views are sorted alphabetically according to their name. The options in the Sort panel of the Editor Preferences dialog let you change this.

Unsorted	Leaves items in these lists unsorted. For views which list definitions, choosing this option lists definitions in the order in which they appear in the source code.
By Name	Sort buffers according to their names. This is the default setting.
By Package	Sort buffers according to the buffer package.
Ву Туре	Sorts items according to the type of each item. This sorts different types of definition, or different types of buffer, as appropriate.

10.7.2 Displaying package information

As with many other tools, you can configure the way package names are displayed in the editor. Because of the nature of this tool, you need to be a little more aware of the precise nature of these commands in order to avoid confusion. This information can be configured using the Package panel of the Editor Preferences dialog.

Click **Show Package Names** to toggle display of package names in the main areas of the buffers, definitions and changed definitions views.

Type a package name into the text field to change the current package in the editor. Click the v button to confirm the package name, or the button v to display a completion list for all package names. Note that this does *not* change the package currently displayed; it merely changes the editor's notion of "where" it is in the environment, and this in turn affects the way symbols are printed in the buffers, definitions and changed definitions views.

By default, the current package is CL-USER.

10.8 Getting started with the editor

This section deals with some of the most basic commands available in the editor. It describes how to perform simple file management, how to move around a buffer, and tells you about some other more general commands available.

10.8.1 Opening, saving and printing files

When you first start up the editor, the first thing you must do is open a file.

Use file extensions .lisp or .lsp for Common Lisp files. The editor recognizes these extensions and places the buffer in Lisp mode. Lisp mode provides special features for use in Lisp editing.

You can create a new Lisp buffer by choosing **File > New** or clicking on **D**. The new file is automatically in Lisp mode, and the buffer is called "Unnamed". As soon as you try to save this buffer, the editor prompts you for a filename.

As you have already seen, you can open an existing file by choosing File > **Open** or clicking on A dialog appears from which you can select a file to edit.

To save a file, choose File > Save or click on . If the file has not been saved before (that is, if you created the file by choosing File > New and this is the first

time you have saved the file), you are prompted for a directory and a filename.

You can also save a file by using the keyboard command Ctrl+x Ctrl+s.

If you want to make a copy of the file (save the file under a different name) choose **File > Save As** and specify a name in the dialog that appears.

Choose File > Revert to revert back to the last saved version of the file. This replaces the contents of the current buffer with the version of that file which was last saved on disk. This command can be useful if you make a number of changes which you want to lose.

As well as saving whole files to disk, you can save any part of a file to disk under a different filename. To do this:

- 1. Select a region of text by clicking and holding down the select mouse button, and dragging the pointer across the region of text you want to save. The text is highlighted as you drag the pointer across it.
- 2. With the text still highlighted, choose File > Save Region As.
- **3.** In the echo area, specify the name of a file to save the selected text to.

Note that the selected text is *copied* into the new file, rather than moved; it is still available in the original buffer.

To find out more about selecting regions of text, see Section 10.11.1 on page 123. To find out more about operating on regions of text, see Section 10.13 on page 129.

To print the file in the current buffer to your default printer, choose File > Print. The printer on which your hard copy appears can be changed or configured by using the standard Windows Control Panel

10.8.2 Moving around files

This section describes how you can move the cursor around the buffer. There are a variety of commands, allowing you to move sideways, up, or down by one character, or by a number of characters.

To move directly to any point in the buffer, position the pointer and click the left mouse button. If necessary, use the scroll bars to reveal sections of the buffer which are not visible in the window.

You can either use the arrow keys, or the keyboard commands shown below to move the cursor in any direction by one character.



Figure 10.6 Moving the cursor by one character

The keyboard commands below move to the beginning or end of the line, or the top or bottom of the buffer.





Press Ctrl+v or the Page Down key to scroll down one screenful of text.

Press Alt+v or the Page Up key to scroll up one screenful of text.

You should ensure that you learn all the keyboard commands described above, since they make navigation in a buffer much easier.

10.8.3 Inserting and deleting text

The editor provides a sophisticated range of commands for cutting text which are described in Section 10.11 on page 123. However, the two basic commands for deleting text which you should remember are as follows:

- To erase the previous character, use the Backspace key.
- To erase the next character, use Ctrl+D or the Delete key.

You can insert text into a buffer by typing characters, or by pasting (see Section 10.11 on page 123).

By default, when inserting text into a buffer, any characters to the right of the cursor are moved further to the right. If you wish to write over these characters, rather than preserve them, press the Insert key. To return to the default behavior again, just press the Insert key once more.

To insert the contents of one file into another, choose File > Insert. A dialog appears so that you can choose a file to insert, and this is then inserted into the current buffer, starting from the current position of the cursor.

10.8.4 Using several buffers

As mentioned above, you can have as many buffers open at once as you like. Repeated use of **File > Open** or Ctrl+x Ctrl+F just creates extra buffers.

Because the editor can only display one buffer at a time, you can use either menu commands or keyboard commands to swap between buffers.

Each item in the **History > Items** submenu is an open buffer. To swap to a given buffer, choose it from the menu, and it is displayed in the editor window.

Alternatively, click on the Buffers tab to swap to the buffers view; see Section 10.2 on page 112 for details.

To use the keyboard, type Ctrl+x B. You are prompted for the name of the buffer you wish to display. The last buffer you displayed is chosen by default, and is listed in the echo area in brackets, as shown below.

Select Buffer: (test.lisp):

To swap to the buffer shown in brackets, just press Return. To swap to another buffer, type in the name of that buffer. Remeber that Tab completion can help.

To close the buffer that is currently displayed, choose File > Close, or press Ctrl+x κ .

- If you use File > Close, the current buffer is closed.
- If you use Ctrl+x κ, you can close *any* buffer, not just the current one. Type a buffer name in the echo area, or press Return to close the current buffer.

Note: If you attempt to close any buffer which you have changed but not yet saved, a dialog appears, giving you the opportunity to cancel the operation.

To save all the buffers in the editor, choose File > Save All. A dialog appears which lists each modified buffer. By default, each buffer is selected, indicating that it is to be saved. If there are any buffers that you do not want to save, deselect them by clicking on them. The dialog has four buttons, as follows:

Click **Yes** to save the selected buffers.

Click All to save all the listed buffers.

Click No to save none of the listed buffers.

Click Cancel to cancel the operation.

This dialog is also displayed if there are any unsaved files when you exit the environment.

Sometimes you may find that being able to display only one buffer in the window simply does not give you enough flexibility. For instance, you may have several buffers open, and you may want to look at two different buffers at once. Or you may have a very large buffer, and want to look at the beginning and end of it at the same time.

You can do any of these by creating a new editor window. Choose Tools > Clone or press Ctrl+x 2. This creates a copy of your original editor. The new editor displays the same buffer as the original one.

- If you want to look at two different sections of this buffer at once, simply move to the section that you want to look at in one of the editors.
- If you want to look at a different buffer, use the **History > Items** submenu or the keyboard commands described above to switch buffers.

Changes made to a buffer are automatically reflected across all editor windows—the buffer may be displayed in two different windows, but there is still only one buffer. This means that it is impossible to save two different versions of the same file on disk.

10.9 Other essential commands

Finally, there are three basic functions which you should add to your stock of familiar commands.

10.9.1 Aborting commands

To abort any command which requires you to type information at the echo area, type Ctrl+G at any point up to where you would normally press Return. For instance, if you type Ctrl+x Ctrl+F in order to open a file, and then decide against it, type Ctrl+G instead of specifying a filename.

10.9.2 Undoing commands

If you choose Edit > Undo, or press Ctrl+_, the last editor action performed is undone. Successive use of Edit > Undo revokes more actions (rather than undoing the last Undo command, as is the case with many other editors). Thus, to undo the last five words typed, press Ctrl+_ five times.

10.9.3 Repeating commands

To perform the same command *n* times, type Ctrl+u *n* followed by the command you want to perform.

For instance, to move forward 10 characters, type Ctrl+U 10 Ctrl+F.

10.10 Cutting, copying and pasting using the clipboard

The editor provides the standard methods of cutting, copying and pasting text using the clipboard. To select a region of text, click and hold down the select button, and drag the pointer across the region you want to select: the text is highlighted as you select it.

Choose Edit > Select All to select all the text in the buffer, and Edit > Deselect All if you want to deselect it.

Once you have selected a region use either of the following commands:

Choose **Edit > Copy** to copy the region to the clipboard. This leaves the selected region unchanged in the editor buffer.

Choose **Edit > Cut** to delete the region from the current buffer, and place it in the Common LispWorks clipboard. This removes the selected region from the buffer.

Choose **Edit > Paste** to copy text from the clipboard into the current buffer. The text is placed at the current cursor position.

These commands are also available from a popup menu that you can display by using the alternate select gesture (usually clicking the right mouse button) over the editor window.

The editor also provides a much more sophisticated system for cutting, copying and pasting text, as described below.

10.11 Cutting, copying and pasting using the kill ring

The editor provides a sophisticated range of commands for cutting or copying text onto a special kind of clipboard, known as the *kill ring*, and then pasting that text back into your editor later on. There are three steps in the process, as follows:

- Marking a region of text.
- Cutting or copying the text in that region to place it in the kill ring.
- Pasting the text from the kill ring back into a buffer.

10.11.1 Marking the region

First of all, you need to mark a region of text in the current buffer which you want to transfer into the kill ring. There are two ways that you can do this:

• Select the text you want to copy or cut using the mouse. Click and hold down the Select mouse button, and drag the pointer across the region you want to mark.

The selected text is highlighted using a bold font.

• Using keyboard commands

To mark the region with the keyboard, place the cursor at the beginning of the text you want to mark, press Ctrl+Space, and move the cursor to the end of the region you want to mark, *using keyboard commands to do so*. Unlike marking with the mouse, this does not highlight the region.

Because the editor does not highlight the marked region when you use keyboard commands, a useful command to remember is Ctrl+x Ctrl+x. Pressing this exchanges the current cursor position with the start of the marked region, highlighting the region in the process. Press Ctrl+x Ctrl+x a second time to return the cursor to its original position, but leave the region marked.

Press Ctrl+G to turn off the highlighting in a region.

10.11.2 Cutting or copying text

Once you have marked the region, you need to transfer the text to the kill ring by either cutting or copying it.

Cutting text moves it from the current buffer into the kill ring, and deletes it from the current buffer, whereas copying just places a copy of the text in the kill ring.

- Choose Edit > Cut or press Ctrl+w to cut the text.
- Choose Edit > Copy or press Alt+w to copy the text.

Notice that these commands transfer the selected text to the Common LispWorks clipboard as well as the kill ring. This is so that the selected text can be transferred into other tools, or even into other applications.

10.11.3 Pasting text

Once you have an item in the kill ring, you can paste it back into a buffer as many times as you like.

• Press Ctrl+Y to paste the text in the kill ring back into the buffer.

Note that you must use the keyboard command if you wish to paste the item that is in the kill ring (as opposed to the item in the Common LispWorks clipboard).

With many editors you can only do this with one item at a time. The clipboard is only able to contain one item, and so it is the only one available for pasting back into the text.

However, the kill ring allows you to keep many items. Any of these items can be pasted back into your document at any time. Every time you cut or copy something, it is added to the kill ring, so you accumulate more items in the kill ring as your session progresses.

Consider the following example. In Figure 10.8, the kill ring contains three items; the words factorial, function and macro respectively.



Figure 10.8 Kill ring with three items

First, the word factorial was cut from the current buffer (this would remove it from the buffer). Next, the word function was copied (which would leave it in the buffer but add a copy of it to the kill ring), and lastly, the word macro was cut.

Note the concept of the kill ring rotating (this is why it is known as a ring). Every time a new item is added (at the top, in these figures), the others are all shunted around in a counter-clockwise direction.

Whenever you perform a paste, the current item in the kill ring—the word macro in this case—is copied back into the buffer wherever the cursor currently is. Note that the current item is not removed from the kill ring.



Figure 10.9 Pasting from the kill ring

What you have seen so far does exactly the same thing as the standard clipboard. True, all three items have been kept in the kill ring, but they are of no use if you cannot actually get at them.

The command to do this is Alt+Y. This rotates the kill ring in the opposite direction—thus making the previous item the current one—and pastes it into the buffer in place of the item just pasted. In Figure 10.9, the word macro would be replaced with the word function.

You can use Alt+Y as many times as you like. For instance, if you actually wanted to paste the word factorial in the document, pressing Alt+Y would replace the word function with the word factorial.



Figure 10.10 Rotating the kill ring

If you pressed Alt+Y a third time, the kill ring would have rotated completely, and macro would have been the current item once again.

Note: You can never use Alt+Y without having used Ctrl+Y immediately beforehand.

Here is a summary of the way Ctrl+Y and Alt+Y work:

- Ctrl+Y pastes the current item in the kill ring into the buffer.
- Alt+Y rotates the kill ring back one place, and then pastes the current item into the buffer, replacing the previously pasted item.

10.12 Searching and replacing text

The editor provides a wide range of facilities to search for and replace text. The examples below introduce you to the basic principles; please refer to the *Editor User Guide* for a complete description of the facilities available.

10.12.1 Searching for text

The simplest way of searching for text in a buffer is to use the commands available in the menu bar.

Choose **Edit > Find** to search for text in the current buffer. Type a string to search for in the dialog that appears, and click **OK**. The cursor is placed immediately after the next occurrence in the current buffer of the string you specified. Choose **Edit > Find Next** to find the next occurrence of the same string.

10.12.2 Incremental searches

Press Ctrl+s to perform an *incremental search* (in which every character you type further refines the search). A prompt appears in the echo area, asking you to type a string to search for. As soon as you start typing, the search commences.

Consider the following example: You want to search for the word "defmacro" in a file.

1. Select Edit > Find... or press Ctrl+s

The following prompt appears in the echo area.

I-Search:

2. Type the letter a.

The prompt in the echo area changes to

I-Search: d

The cursor moves to the first occurrence of "d" after its current position.

3. Type the letter e.

The prompt in the echo area changes to

I-Search: de

The cursor moves to the first occurrence of "de".

4. Type the letter **f**.

The prompt in the echo area changes to

I-Search: def

The cursor moves to the first occurrence of "def".

This continues until you stop typing, or until the editor fails to find the string you have typed in the current buffer. If at any point this does occur, the prompt in the echo area changes to reflect this. For instance, if your file contains the word "defun" but not the word "defmacro", the prompt changes to

Failing I-Search: defm

as soon as you type m.

10.12.3 Replacing text

You can search for text and replace it with other text using the **Edit > Replace**. In the echo area, you are prompted to supply the text to be found and the text to replace it. In addition, whenever that text is found, you are asked whether or not to replace that occurrence of it. Note that this type of searching is *not* incremental.

For instance, assume you wanted to replace every occurrence of "equal" to "equalp".

1. Choose Edit > Replace.

The following appears in the echo area:

Query Replace string:

2. Type equal and press Return.

The echo area now shows the following:

Replace "equal" with:

3. Type equalp and press Return.

At every occurrence of "equal" after the current cursor position, the following message is printed in the echo area:

Replace equal with equalp?

- If you want to replace this occurrence, type y.
- If you do not want to replace this occurrence, type n.
- If you want to abandon the operation altogether, press Esc.

Other options are also available. See the *LispWorks Guide to the Editor* for more details. You can also immediately use the editor command **Help on Parse** to see a list of these options. (Type ? at the prompt in the echo area to invoke **Help on Parse**).

Note: Both **Edit > Find** and **Edit > Replace** start searching from the current position in the buffer. When **Edit > Find** reaches the end of the buffer, it starts again at the beginning, but **Edit > Replace** stops. To replace all occurrences of a text string in the buffer, you must ensure you are at the top when you begin.

10.13 Using Lisp-specific commands

One of the main benefits of using the built-in editor is the large number of keyboard and menu commands available which can work directly on Lisp code. As well as providing editing facilities which work intelligently in a buffer containing Lisp code, there are commands which let you load, evaluate or compile the code in any part of a buffer easily and quickly.

In addition, a high degree of integration exists between other Common LispWorks tools and the editor. This allows you, for example, to find the source code definition of an object being examined in a browser, or to flag symbols in editor buffers for specific actions, such as tracing or lambda list printing. This section provides an introduction to the Lisp-specific facilities that are available using menu commands. For a full description of the extended editor commands, please refer to the *Editor User Guide*.

All of the commands described below are available in the editor's **Works > Buffers**, **Works > Definitions**, and **Works > Expression** menus. They operate on the current buffers, definitions, or expression, the choice of which is affected by the current view. In the Text view, the Buffers menu applies to the currently visible buffer, the Definitions menu applies to the highlighted definitions or to the definition under the cursor and the Expression menu depends on the exact location of the cursor, as described below. In the Buffers view, the Buffers menu applies to all the selected buffers and the Definitions and Expression menus are not available. In the Definitions, Changed Definitions and Find Definitions views, the Definitions menu applies to the selected definitions and the Buffers and Expression menus are not available.

An expression is the symbol over which the cursor is positioned (or the one immediately before it if it is not on a symbol), and a definition is the definition in which that symbol occurs.

```
(defun test ()
(test2))
```

In the function shown above, if the cursor were placed on the letter "e" of test2, the expression would be the symbol test2, and the definition would be test.

In this section, the term "current definition(s)" is used to denote either the definition under the cursor (in the text view), or the selected definitions (in either the definitions or the find definitions views).

10.13.1 Evaluating code

When you are editing Lisp code, you may want to evaluate part or all of the buffer in order to test the code. The easiest way to do this is using menu commands, although there are keyboard commands which allow you to evaluate Lisp in the editor as well.

There are three menu commands which allow you to evaluate Lisp in the current buffer. Choose **Works > Buffers > Evaluate** to evaluate all the code in the current buffer. If you are in the buffers view, then this command evaluates the code in all the selected buffers.

Choose **Works > Expression > Evaluate Region** to evaluate the Lisp code in the current region. You must make sure you have marked a region before choosing this command; see Section 10.11.1 on page 123. Whether you use the mouse or keyboard commands to mark a region does not matter. If you have a few Lisp forms that you want to evaluate, but do not want to evaluate the whole buffer, you should use this command.

Choose **Works > Definitions > Evaluate** to evaluate the definition in which the cursor currently lies. This is a little like evaluating the marked region, except that only the current definition is evaluated, whereas working with a marked region lets you evaluate several. This command is useful if you have a single function in the current buffer which you want to test without taking the time to evaluate the whole buffer or mark a region.

In the definitions or find definitions views, this command evaluates the code for all the selected definitions, allowing you to evaluate code for a number of definitions with one command.

To load the code for any file into the environment (even if it is a file not currently loaded into the editor), choose File > Load. You are prompted for a pathname in the echo area.

10.13.2 Compiling code

You can also compile Lisp code in an editor buffer in much the same way that you can evaluate it.

Choose Works > Buffers > Compile to compile all the code in the current buffer.

Choose **Works > Expression > Compile Region** to compile the Lisp code in the current region.

Choose **Works > Definitions > Compile** to compile the definition in which the cursor currently lies.

During compilation, the editor window temporarily displays compiler output in an output window. Once compilation has finished, press the space bar to display the current buffer once again. You can view this output window at any time by clicking the Output tab of the editor.

To compile the code for any file, even if it is not loaded into the editor, choose **File > Compile**. You are prompted for a filename in the echo area. If you wish to load the compiled file into the environment as well, choose **File > Compile and Load**.

10.13.3 Tracing symbols and functions

A wide variety of tracing operations are available in the **Works > Buffers**, **Works > Definitions** and **Works > Expression** menus. The scope of each operation depends on which menu the command is chosen from.

Choose **Trace** from either the **Works > Buffers**, **Works > Definitions** or **Works > Expression** menus to display a menu of trace commands that you can apply to the current region or expression, or the currently selected buffers or definitions, as appropriate. See Section 3.10 on page 36 for full details. Note that you can select several items in the buffers, definitions and find definitions views.

See Section 3.10 on page 36 for full details of the tracing facilities available in the editor.

10.13.4 Packages

Each buffer has a package associated with it, known as the primary package. If there is an in-package form in a file when it is first read in, this is taken to specify the primary package; otherwise it is taken to be CL-USER. If the primary package has not already been seen by the environment, you are prompted for its creation. The primary package is shown in the message line at the bottom of the editor window.

When evaluating parts of the current buffer (as opposed to all of it), the environment only uses the buffer's primary package if no in-package form is found in the selected regions. It tries to find the current package within the section of code which is being evaluated. This is done so that any occurrences of in-package in the buffer (other than on the first occurence) can be allowed for—if the current package was always assumed to be the primary package of the current buffer, many sections of code would evaluate wrongly.
This means that you do not have to worry about setting the package explicitly before evaluating part of a buffer.

10.13.5 Indentation of forms

The editor provides facilities for indenting your code to help you see its structure. These facilities are available only in Lisp mode.

Alt+Ctrl+Q indents the current Lisp form, and the Tab key indents a single line.

10.13.6 Other facilities

A number of other Lisp-specific facilities are available using the menus in the editor.

Choose **Works > Definitions > Undefine** to remove the selected definitions from the environment. Similarly, choose **Works > Buffers > Undefine** to remove the definitions in the selected buffers. By selecting items in the buffers, definitions or find definitions views, you have precise control over the definitions which can be removed with one command.

Choose **Works > Definitions > Generic Function** to describe the current definition in a generic function browser. See Chapter 12, "The Generic Function Browser" for more details.

Standard action commands can be found on the **Works > Expression** menu, allowing you to perform a number of operations on the current expression. See Section 3.8 on page 31 for full details.

Choose **Works > Expression > Arguments** to print the lambda list of the current expression in the echo area, if it is a function, generic function or method. This is the same as using the keyboard command Alt+=, except that the current expression is automatically used.

Choose **Works > Expression > Value** to display the value of the current expression in the echo area.

Choose **Works > Expression > Macroexpand** to macroexpand the current form. The macroexpansion is printed in an output window, in the same way that

compilation output is shown. Click on the Output tab to redisplay the output at any time.

Choose **Works > Expression > Walk** to recursively macroexpand the current form.

10.14 Help with editing

Two help commands are available which are specific to the editor and any tools which use editor windows.

Choose **Help > Editing > Key to Command** and type a key sequence to display a description of the function it is bound to, if any.

Choose **Help > Editing > Command to Key** and supply an editor command name to see the key sequence it is bound to, if any.

The Function Call Browser

11.1 Introduction

The function call browser gives you a way to view any function in the environment together with the functions that call it or the functions it calls. It has three views available.

- The *Called By* view allows you to examine a graph of the functions which call the function being browsed. This is the default view.
- The *Calls Into* view allows you to examine a graph of the functions which are called by the function being browsed.
- The *Text* view lets you see immediate callers and callees of the browsed function using lists rather than a graph.

To create a function call browser, choose **Tools > Function Call Browser** from the LispWorks podium. Alternatively, select a function in another tool, and choose **Browse** from the appropriate actions menu to browse the selected function in the function call browser. Finally, in an editor executing Alt-x List Callers Or Alt-x List Callers calls up a function call browser on the function over which the cursor is placed.

11.2 Examining functions using the graph view

The *Called By* view is the default view in the function call browser. When in the graph view, the function call browser appears as in Figure 11.1.

🛃 Function Call Browser 1	_ 🗆 ×
Eunction: DO-NUMBERS	🗸 🗙 🚱
Text Called By Calls Into	
DEMO DO-NUMBERS	
	▼ ▶
Function <u>D</u> escription >>	
Ready.	

Figure 11.1 Viewing functions using the "Called By" view

In this view, the function call browser has three areas.

11.2.1 Function area

The *Function area* is used to enter the name of the function you want to examine. As with similar areas in other tools, you can press the Tab key at any point while you type the function to either complete the function name or (if there is more than one possible completion) display a dialog listing all the functions available in the current package.

11.2.2 Graph area

A graph of all the callers of the function is displayed in the *Graph area*. Note that if the toggle-source-debugging function has not been used to set source debugging to t (it's default setting), the graph view has no information to display.

The graph area of the *Calls Into* view is the same, other than the fact that the graph displayed is of the functions called by the function being browsed.

The generic facilities available to all graph views throughout the environment are available here; see Chapter 5, "Manipulating Graphs" for details.

11.2.3 Echo area

The echo area of the function call browser is similar to the echo area of the podium. It displays messages concerning the function call browser.

11.3 The Function Description button

Clicking on **Function Description >>** changes the view of the function call browser to include more information on the function being browsed. The browser appears as in Figure 11.2

& Function Call Browser 1		
Eunction: FACT	< 🗙 😵	
Text Called By Calls Into		
Called by:	Calls into:	
	* 1- FACT ZEROP	
Name: FACT Function: # <function 20410d12="" fact=""> Lambda List: (N) Source Files: c:\TEMP\fact.lisp</function>		
Documentation:		
	<u> </u>	
	V	
Function <u>D</u> escription <<		
Ready.		

Figure 11.2 The function call browser in function description mode

Two further panes appear, a function description area, and a documentation area.

11.3.1 Function description area

The *Function Description area* gives a description of the function selected in the Graph area, or, if nothing is selected, the current function (as displayed in the Function area). The following items of information are displayed:

Name	The name of the function.
Function	The function object.

Lambda List The lambda list of the function.

Source Files The source file in which the function is defined, if any.

You can operate on any of the items in this area using the commands in the **Works > Description** menu. This contains the standard actions described in Section 3.8 on page 31.

11.3.2 Documentation area

The *Documentation area* shows the documentation for the function selected in the graph area, if there is any. If no function is selected, the documentation for the current function is shown, if there is any.

11.4 Examining functions using the text view

Click on the *Text* tab to see a textual display of the callees and callers of a function. This view has the advantage that both callees and callers can be seen simultaneously. It is very similar to the text view in the class browser, as described in Section 7.1.4 on page 65. When in the text view, the function call browser appears as shown in Figure 11.3.

& Function Call Browser 1	
Eunction: FACT	🖌 🗙 🐉
Text Called By Calls Into	
Called by:	Calls into:
FACT	* FACT ZEROP
Function <u>D</u> escription >>	
Ready.	

Figure 11.3 Viewing functions using the text view

When in this view, four areas are available.

11.4.1 Function area

The name of the current function is shown here. This area is exactly the same as the Function area in the graph view; see Section 11.2.1 on page 136 for details.

11.4.2 Called By area

The *Called By* area lists those functions which the current function calls.

To make any function in this list the current function, double-click on it.

11.4.3 Calls Into area

The *Calls Into* area lists those functions which call the current function.

To make any function in this list the current function, double-click on it.

11.4.4 Echo area

The echo area of the function call browser is similar to the echo area of the podium. It displays messages concerning the function call browser.

Clicking on Function Description >> calls up two more areas:

11.4.5 Function description area

This gives a description of the function selected in the Callees or Callers areas, or the current function if nothing is selected in either of these areas. The same information is listed as in the graph view; see Section 11.3.1 on page 138 for details.

11.4.6 Documentation area

This area displays documentation for the function selected in either the Callees or Callers areas, or the current function.

11.5 Configuring the function call browser

The function call browser can be configured using the preferences dialog. Select **Tools > Preferences** in the function call browser to call up the dialog, which is shown in Figure 11.4

🖉 Function Call Browser 1 Preferences 🛛 🗙	
General Called By I Sort ○ Unsorted ○ By Name ○ By Package	Layout Calls Into Layout Package
	<u>OK</u> <u>C</u> ancel <u>Apply</u>

Figure 11.4 The function call preferences dialog.

11.5.1 Sorting entries

Entries in the Graph area, Callers are and Callees area can be sorted in a number of ways.

Choose **By Name** to sort entries according to the function name. This is the default setting.

Choose By Package to sort functions according to their package.

Choose Unsorted to leave functions unsorted.

11.5.2 Displaying package information

As with other tools, you can configure the way package names are displayed in the function call browser.

Choose **Show Package Names** to turn the display of package names in the Graph, Callers, Callees and Description areas on and off.

Enter a different package name in the **Package** text input box to change the current package in the function call browser. By default, the current package is CL-USER.

11.6 Configuring graph displays

The preferences dialog can also be used to configure how the function call browser displays graphical information in the *Called By* and *Calls Into* views. Click on the *Called By Layout* or *Calls Into Layout* tab in the preferences dialog. Both views perform the same operations on the relevant function call browser view.

🛃 Function Call Br	owser 1 Preferences	×
General Called By Layout C Left to Right I Deft to Left I Deft Down C Bottom Up	Layout Calls Into Layout Analysis Analysis Calls Into Layout Analysis Analysis Analysis Calls Into Layout Analysis Analy	T
	<u> </u>	Apply

Figure 11.5 A layout view in the preferences dialog

11.6.1 Graph layout settings

The layout radio buttons are used to set the direction in which the graph is displayed. The default setting is **Left to Right**.

11.6.2 Graph expansion settings

The Max. Expansion settings determine how much of the graph to display. The default depth value is 2—this ensures that only functions that directly call (or are directly called by) are shown in the graph. If this value were set to 3, for example, then functions that call a function that calls the function being browsed would also be displayed.

The breadth value has a default value of 40, and sets how many functions are displayed at each level of the graph.

11.6.3 Plan mode settings

The **Rotation** checkbox determines whether the graph layout can be rotated when in plan mode. By default it is checked.

You can enter plan mode when displaying a graph by holding down your right mouse button and selecting **Enter Plan Mode** from the pop-up menu. If rotation is enabled, you can rotate the plan by holding down the Shift key and moving the mouse left or right.

11.7 Performing operations on functions

A number of operations can be performed on functions selected in the Text area (when in the text view) or in the Callees or Callers areas, or on the current function (when there are no functions selected elsewhere).

The **Works > Function** menu gives you access to the standard actions described in Section 3.8 on page 31.

The **Works > Function > Trace** submenu gives you the ability to trace and untrace the functions selected in the Text, Callers or Callees area.

The Generic Function Browser

The generic function browser allows you to examine the generic functions in the environment, together with any methods that have been defined on them. It has two views which allow you to browse different types of information:

- The methods view, which shows you a description of the generic function and the methods defined on it. This is the default view.
- The method combinations view, which lets you examine the list of method combinations for any generic function.

To create a generic function browser, choose **Tools > Generic Function Browser**. Alternatively, if the current object in a tool is a generic function or method, choose the **Generic Function** standard action command from the appropriate menu, or use the editor command **Alt+X Describe Generic Function**, to display it in the generic function browser.

12.1 Examining information about methods

When the generic function browser is first displayed, the default view is the methods view. You can also choose it explicitly by clicking on the Methods tab of the generic function browser.

The methods view is shown in Figure 12.1 below.

Ceneric Function Browser I	×
Function: CAPI:SIMPLE-PANE-ENABLED	?
	2.1
Methods Method Combinations	
Filter: Matches: 2	
(METHOD CAPI:SIMPLE-PANE-ENABLED (CAPI:SIMPLE-PANE))	
(METHOD CAPI:SIMPLE-PANE-ENABLED (CAPI::TOOLBAR-OBJECT	9 (
	_
Description:	_
Generic Function: CAPI:SIMPLE-PANE-ENABLED	
Class: STANDARD-GENERIC-FUNCTION	
Lambda List: (CAPI::TOOLBAR-OBJECT)	
Combination: STANDARD	
Method Close:	
Method Signeture:	
L	
	_

Figure 12.1 Generic function browser

The methods view has four main sections, described below.

12.1.1 Function box

The Function box shows the name of the generic function you are examining. To browse a generic function, type its name into the Function area, or paste it in from another tool in one of two ways:

- Choose Edit > Copy or the standard action command Copy in another tool to copy the generic function to the clipboard, then choose Edit > Paste in the generic function browser to transfer the generic function in.
- Choose the standard action command **Generic Function** in the other tool to display the generic function in the generic function browser in one action.

When typing in the name of a function, you can click *here are a complete the series of the function, or display a list of possible completions from which you can select the function you want to browse.*

Note: You can use **Edit > Paste** to paste in a generic function, even if the Common LispWorks clipboard currently contains the string representation of the function, rather than the function itself. This lets you copy in generic functions from other applications, as well as from the environment. See Section 3.4.2 on page 25 for a complete description of the way the Common LispWorks clipboard operates.

You can operate on the current generic function using the commands in the generic function browser's **Works > Function** menu. See Section 12.1.5 on page 149 for details.

12.1.2 Filter box

The Filter box lets you restrict the information displayed in the list of methods, so that only the methods you are interested in are displayed. See Section 3.12 on page 38 for details about how to use the Filter box.

12.1.3 List of methods

The list of methods provides a list of the methods defined on the generic function.

- Selecting a method in this list displays its description in the Description list.
- Double-clicking on a method displays its source code definition in the editor, if it is available.

The number of items listed in the list of methods is printed in the Matches box.

You can operate on any number of selected methods in this area using the commands in the generic function browser's **Works > Methods** menu. See Section 12.1.5 on page 149 for details.

12.1.4 Description list

The Description list shows a description of the method selected in the list of methods. The following information is listed:

Generic Function

	The name of the generic function on which the method is defined.
Class	The class of the generic function on which the method is defined.
Lambda List	The lambda list of the generic function.
Combination	The class of method combination for the generic func- tion.
Method	The method object that is selected in the list of methods.
Method Class	The method class of which the selected method is an instance.

Method Signature

The signature for the selected method.

To operate on any of the items displayed in this area, select them and choose a command from the **Works > Description** menu. This menu contains the standard action commands described in Section 3.8 on page 31. You can operate on more than one item at once by making a multiple selection in this area.

12.1.5 Performing operations on the current function or selected methods

You can use the **Works > Function** and **Works > Methods** menus to access commands that operate on the current generic function or the selected methods. These commands are similar to commands available in other tools, and so you should find them familiar.

The following commands are available from either the Works > Function or Works > Methods menus:

The standard action commands let you perform a number of operations on the selected methods or the current function. For details on the commands available, see Section 3.8 on page 31.

Choose **Undefine** to undefine the current generic function or the selected methods so that they are no longer available in the environment. Choosing **Undefine** on a method undefines the method function and removes it from the methods on the generic function. However, the generic function can still be called with its different method selection.

The **Trace** submenu gives you the ability to trace and untrace the current generic function or the selected methods. See Section 3.10 on page 36 for details about the commands available in this submenu.

12.2 Examining information about combined methods

The method combinations view lets you examine information about the combined methods of the current generic function. On supplying a signature to the generic function browser, the combined methods of the generic function are displayed together with the arguments that match that method combination point.

Method combinations show you the calling order of methods defined on a generic function. They use the class precedence lists of the classes on which the methods of a generic function operate. Being able to view these combinations gives you a simple way of seeing how before, after, and around methods are used in a particular generic function.

You can display this view by clicking the Method Combinations tab. The method combinations view is shown in Figure 12.2 below.

🛃 Generic Function B	Browser 1	
Eunction: CAPI:SIMPLE-P/	ANE-ENABLED 🗸 🔀 🦻	
Methods Method Combinations Method Combination Arguments Types: (CAPI::TOOLBAR-OBJECT) Signatures CALL-METHOD (METHOD CAPI: SIMPLE-PANE-ENABLED (CAPI::TOOLBAR-OBJEC)		
Description:		
Generic Function: Class: Lambda List: Combination: Method: Method Class: Method Signature:	CAPI:SIMPLE-PANE-ENABLED STANDARD-GENERIC-FUNCTION (CAPI::TOOLBAR-OBJECT) STANDARD	

Figure 12.2 Generic function browser displaying method combinations

The method combinations view has a number of main sections, described below.

12.2.1 Function box

As with the methods view, the name of the generic function being browsed is shown here. See Section 12.1.1 on page 146 for details.

12.2.2 Signatures button

Click **Signatures** to display the Method Signature dialog shown in Figure 12.3. This dialog lists the signatures for the methods defined on the current generic function. The signatures of a method are the types of the arguments that each method defined takes.

Method Signatures	×
Restricted Class:	
NIL	✓ × ≫
Signatures	
<u>F</u> ilter:	🖌 🗙 Matches: 3
(CAPI:SIMPLE-PANE) (CAPI::TOOLBAR) (CAPI::TOOLBAR-OBJECT)	
	<u>D</u> K <u>C</u> ancel Apply

Figure 12.3 Method Signatures dialog

To list the method combinations of any defined method in the generic function browser, select its signature from the list in the Signatures panel and click **OK**.

You can also manipulate the signatures displayed using the Restricted Class panel. See Section 12.2.6 on page 153 for details.

12.2.3 Arguments types box

The Arguments types box is used to specify a signature, in order to see the method combinations. You can specify a signature by either

- Choosing a signature using the Method Signatures dialog, as described in Section 12.2.2.
- Typing the signature list in directly and clicking 🗹.

The method combinations for the relevant method are displayed in the list of method combinations.

12.2.4 List of method combinations

The main list in the method combinations view shows method combinations for the signature specified in the Arguments Types box.

- Selecting any method in the list displays its description in the Description list.
- Double-clicking on any method in the list displays its source code definition in the editor, if it is available.

You can operate on any number of selected methods in this area using the commands in the **Works > Methods** menu. See Section 12.1.5 on page 149 for details.

12.2.5 Description list

The Description list displays a description of any method selected in the list of method combinations. The same items of information are shown as in the methods view; see Section 12.1.4.

To operate on any of the items displayed in this area, select them and choose a command from the **Works > Description** menu. This menu contains the standard commands described in Section 3.8 on page 31. You can operate on more than one item at once by making a multiple selection.

12.2.6 Performing operations on signatures

You can manipulate the signatures displayed using the Restricted Class panel of the Method Signatures dialog, shown in Figure 12.3, page 151. You can display this dialog by clicking **Signatures** in the generic function browser.

By default, the generic function browser displays the signatures of all methods defined on the generic function. When there are many methods, or when the distinction between different classes is not clear, this can be confusing.

To simplify the display, you can restrict the signatures to a chosen class and its superclasses. To do this, type the name of the class that you wish to restrict the signatures displayed to into text box in the Restricted Class panel. As with similar text boxes in Common LispWorks, click \checkmark to confirm your choice, to cancel the current setting, and \Im to display a list of possible completions.

Once you have made a choice, only those signatures that belong to the specified class or any of its superclasses are listed in the Signatures panel of the dialog. In situations where there are a large number of complicated signatures, this can simplify the display and add to your understanding of the methods defined.

Be aware of the difference between this approach and the use of the Argument box in the Signatures panel.

• Restricting signatures confines the signatures offered in the dialog by means of the class of the signatures.

Click \times to display the signatures for all methods defined once again.

12.3 Configuring the generic function browser

As with other Common LispWorks tools, choose **Tools > Preferences** to display the Generic Function Browser Preferences dialog.

Using the options in the Sort panel, you can sort the items in the generic function browser as you can in many of the other Common LispWorks tools.

By Name Sorts items alphabetically by name. This is the default setting.

By Package Sorts items alphabetically by package name.

By Method Qualifier

Sorts items by the CLOS qualifier of the method. This groups together before, after, and around methods.

Unsorted Displays items in the order they are defined in.

For more information on sorting items, see Section 3.9.1 on page 35.

You can also set the process package of the generic function browser, and choose to hide package names in the display, using the Package panel. See Section 3.7 on page 29 for full details.

13

The Inspector

The inspector is a tool for examining and changing the contents of any symbol. To create an inspector, choose **Tools > Inspector** from the LispWorks podium.

The inspector has the following areas:

- At the top of the inspector, the tab of the main view shows the name of the object.
- A *Filter box* provides a method of filtering out those parts of an object that you are not interested in.
- A list of components shows the contents of the object.
- A small listener pane can be added to the inspector tool, allowing you to evaluate Common Lisp forms in context, without having to switch back to the main listener itself. Select **Show Listener** from the popup menu.

🛃 Inspector 2	_ 🗆 ×
# <capi:list-panel 2116e684="" [4="" items]=""></capi:list-panel>	Listener >>
List Panel Local Slots	
Eiber	V Matahari 44
CAPI:ACCEPTS-FOCUS-P	T 🔺
CAPI::ACTION-CALLBACK	NIL
CAPI::BACKGROUND	WIN32:COLOR_WINDOW
CAPI::CALLBACK-TYPE	NIL
CAPI::COLOR-REQUIREMENTS	NIL
CAPI::CURSOR	NIL
CAPI::DATA-FUNCTION	IDENTITY
CAPI::DECORATION	#S(CAPI::DECORATION-INFO CA)
CAPI::DO-CACHE	#(#S(CAPI::COLLECTION-ITEM-(
CAPI-INTERNALS:ENABLED	Т
CAPI::EXTEND-CALLBACK	NIL
CAPI::FONT	:DEFAULT-PORT-FONT
CAPI::FOREGROUND	WIN32:COLOR_WINDOWTEXT
CAPI::GEOMETRY-CACHE	# <capi::pane-geometry [112x<="" td=""></capi::pane-geometry>
CAPI::HELP-CALLBACK	NIL
CAPI::HINT-TABLE	(:INTERNAL-MAX-HEIGHT T :IN
CAPI-INTERNALS: HORIZONTAL-SCROLL	Т
CAPI::INITIAL-FOCUS-ITEM	NIL
CAPI-INTERNALS: INTERACTION	:SINGLE-SELECTION
CAPI: INTERFACE	# <capi:interface "container"<="" td=""></capi:interface>
CAPI::INTERNAL-BORDER	NIL
CAPI::ITEMS-CALLBACK	NIL
CAPI-INTERNALS: ITEMS-COUNT-FUNCTION	LENGTH
CAPI-INTERNALS: ITEMS-GET-FUNCTION	SVREF
CAPT::TTEMS-MAP-FUNCTION	CAPT::MAP-TTEMS-VECTOR
CAPI:LIST-PANEL [sorted by name]	//

Figure 13.1 Inspector

Depending on the type of object being inspected, there may be a number of different views available.

13.1 Examining objects

The main list is the most interesting part of the inspector. Each item in this list describes an attribute of the inspected object by displaying its name (the first field in each entry) and the printed representation of its value (the second field). For example, the inspection of a CLOS object yields a list of its slots and their values. The description is called an *inspection*.

1. Type the following Common Lisp form into a listener:

(make-instance 'capi:display-pane :text "My Display Pane")

 Choose Works > Values > Inspect from the listener's menu bar. The object you created is inspected in the inspector.

Arr Inspector 2	_ 🗆 ×
# <capi:display-pane 21174504=""></capi:display-pane>	Listener >>
Display Pane Local Slots	
Filter	Matches: 24
CAPI:ACCEPTS-FOCUS-P	NIL
CAPI::BACKGROUND	NIL
CAPI::COLOR-REQUIREMENTS	NIL
CAPI::CURSOR	NIL
CAPI::DECORATION	NIL
CAPI-INTERNALS:ENABLED	Т
CAPI::FONT	NIL
CAPI:: FOREGROUND	NIL
CAPI::GEOMETRY-CACHE	# <capi::pane-geometry [nilxnil="" at<="" td=""></capi::pane-geometry>
CAPI::HELP-CALLBACK	NIL
CAPI::HINT-TABLE	(:MAX-HEIGHT T :MIN-HEIGHT :TEXT-
CAPI-INTERNALS:HORIZONTAL-SCROLL	NIL
CAPI: INTERFACE	NIL
CAPI::INTERNAL-BORDER	NIL
CAPI::NAME	NIL
CAPI::PARENT	NIL
CAPI::PLIST	NIL
CAPI-INTERNALS:REPRESENTATION	NIL
CAPI::RESOURCE-NAME	NIL
CAPI::SCROLL-CALLBACK	NIL
CAPI-INTERNALS: TEXT	"My Display Pane"
CAPI::UPDATES	NIL
CAPI-INTERNALS:VERTICAL-SCROLL	NIL
CAPI::VISIBLE-BORDER	:DEFAULT
<u> </u>	
Π	
CAPI:DISPLAY-PANE [sorted by name]	

Figure 13.2 Examining an item in the inspector

When inspecting instances of CLOS classes, you can choose to display only those slots which are local to the class. By default, all slots are displayed, including those inherited from superclasses of the class of the inspected object.

3. Click the Local Slots tab.

M Inspector 2	- 🗆 🗙
# <capi:display-pane_21174504></capi:display-pane_21174504>	Listener >>
Display Pane Local Slots	<u> </u>
Filter	Matches: 1
CAPI-INTERNALS:TEXT "My Display Pane"	
DISPLAY-PANE (local slots only) [sorted by name]	

Figure 13.3 Inspector showing local slots

Because the text slot is the only slot defined locally for a display-pane, it is the only one listed.

13.2 Filtering the display

Sometimes an object may contain so many items that the list is confusing. If this happens, use the Filter box to limit the display to only those items you are interested in.

The example below shows you how to create and display an instance of a CAPI object, and then filter the list so that the only slots displayed are those you are interested in:

1. In a listener, create and display a button by typing the following:

(capi:contain (make-instance 'capi:button))

2. Choose Works > Values > Inspect to copy the object into the inspector.

As you can see, this object has a large number of slots, but you can use the filter to screen out everything except the available callbacks.

3. Type callback in the Filter box and press Return.

🛃 Inspector 2		- 🗆 ×
# <capi:button "nil"="" 20f9ef54=""></capi:button>		Listener >>
Button Local Slots		
<u>F</u> ilter: callback	✓ ×	🕻 Matches: 🛛
CAPI::ACTION-CALLBACK CAPI::CALLBACK-TYPE CAPI::EXTEND-CALLBACK CAPI::HELP-CALLBACK CAPI::SCROLL-CALLBACK CAPI::SCROLL-CALLBACK CAPI::SELECTION-CALLBACK	NIL NIL NIL NIL NIL NIL NIL	
0		
CAPI:BUTTON [sorted by name]		

Figure 13.4 Using filters to limit the display in the inspector

You can immediately see that the callback types that have slots in CAPI button objects. The names of the slots, together with their slot values for the object created, are displayed in the inspector.

13.3 Operating upon objects and items

The **Works > Object** and **Works > Slots** submenus can be used to perform a number of different operations on either the object being inspected, or the items currently selected in the main list. The commands available are largely identical in both menus, and so are described together in this section.

13.3.1 Examination operations

The standard action commands are available in both the **Works > Object** and **Works > Slots** menus, allowing you to perform a variety of operations on the current object or any items selected in the list. For full details on the standard action commands, see Section 3.8 on page 31.

13.3.1.1 Example

Consider the following example, where a closure is defined:

```
(let ((test-button (make-instance 'capi:button)))
(defun is-button-enabled ()
     (slot-value test-button 'capi::enabled)))
```

This has defined the function is-button-enabled, which is a closure over the variable test-button, where the value of test-button is an instance of the capi:button class.

- 1. Type the definition of the closure shown above into a listener.
- 2. Choose Works > Values > Inspect.

The inspector examines the symbol is-button-enabled.

- 3. Click on the FUNCTION item to select the closure.
- 4. Choose Works > Slots > Inspect to inspect the selected item.

The closure is inspected.

You can also double-click on an item in the Main area to inspect it.

13.3.2 Inspecting attributes

The Works > Slots > Inspect Attributes option on the Works > Slots menu causes the inspector to view the attributes of the selected slots. This is useful when

inspecting hash tables or alists, since the attributes (keys) might be composite objects themselves.

13.3.3 Tracing operations

The Works > Slots > Trace and Works > Object > Trace submenus provides four options. Slot access on CLOS objects may be traced using these commands.

Break on Read causes a break to the debugger if the slot is accessed for read, either by a defined accessor or by slot-value.

Break on Write causes a break to the debugger if the slot is accessed for write, either by a defined accessor or by *slot-value*.

Break on Access causes a break to the debugger if the slot is accessed for read or write, either by a defined accessor or by slot-value.

Untrace turns off tracing on the selected object or slot.

13.3.4 Manipulation operations

As well as examining objects in the inspector, you can destructively modify the contents of any composite object.

This sort of activity is particularly useful when debugging; you might inspect an object and see that it contains incorrect values. Using the options available you can modify the values in the slots, before continuing execution of a program.

Choose **Works > Slots > Set** to change the value of any selected slots. A dialog appears into which you can type a new value for the items you have selected.

Choose **Works > Slots > Paste** to paste the contents of the clipboard into the currently selected items.

13.3.4.1 Example

This example takes you through the process of creating an object, examining its contents, and then modifying the object.

1. In a listener, define the following function, which examines the contents of an button object to find out whether or not it is enabled:

2. Next, create a button as follows:

(setq button1 (make-instance 'capi:button))

- **3.** Choose **Works > Values > Inspect** in the listener to inspect the button in the inspector.
- 4. In the listener, use the function you just defined to find out whether button1 is enabled.

```
(button-enabled-p button1)
```

This returns t, so, buttons are enabled by default. The next step is to destructively modify button1 so that it is not enabled.

5. Choose Tools > Preferences from the inspector's menu bar.

The Inspector Preferences dialog appears, so that you can change the current package of the inspector.

6. In the text box in the Package panel, replace the default package name with CAPI and click **OK**.

This changes the process package of the inspector to the CAPI package, and the package name disappears from all the slots listed. This makes the display a lot easier to read.

7. In the inspector, type enabled into the Filter box and press Return.

Button objects have a large number of slots, and so it is easier to filter out the slots that you do not want to see than to search through the whole list. After applying the filter, only one slot is listed.

- 8. Select the slot enabled to t.
- 9. Choose Works > Slots > Set...

A dialog appears into which you can type a new value for the slot enabled.

🛃 Inspector 2	×
Enter form (to be evaluated) for	selected items:
ni	
<u>0</u> K	<u>C</u> ancel

Figure 13.5 Entering a new slot value

10. Type nil in the dialog and click on OK.

The Main area reflects the new value of ENABLED.

- **11.** Click on the *intermediate button*. This removes the filter and displays all the slots once again.
- **12.** To confirm that the change happened, type the following in the listener. (You should be able to recall the last command using Alt+P.)

(button-enabled-p button1)

This now returns nil, as expected.

The next part of this example shows you how you can modify the slots of an object by pasting in the contents of the clipboard. By default, the background slot of button objects is nil. This example shows you how to modify the back-ground of button1 so that it is red.

13. Type the following into the listener and then press Return:

"Hello World!"

- 14. Choose Works > Values > Copy to copy the string to the clipboard.
- 15. Select the TEXT slot of button1 in the inspector.
- 16. Choose Works > Slots > Paste to paste the "Hello World!" string into the text slot of button1.

This sets the text slot of button1 to the string.

You can confirm the effect of this change by displaying the button object. To do this, transfer it back into the listener and display it as follows: 17. Choose Works > Values > Listen.

This transfers the button object back into the listener. As feedback, the string representation of the object is printed in the listener above the current prompt. The object is automatically transferred to the * variable so that it can be operated on.

18. In the listener, type the following:

(capi:contain *)

This displays a window containing the button object. Note that the text now reads "Hello World!", as you would expect. Note further that you cannot click on the button; it is not enabled. This is because you modified the setting of the enabled slot in the earlier part of this example.

You can also use **Works > Slots> Copy** to copy this value to other objects in the inspector, or even to other tools.

13.4 Configuring the inspector

The Inspector Preferences dialog lists a number of different display options in its default general view. These include the standard options for sorting items in the main list and displaying package information, together with some additional options specific to the inspector.

Market Inspector 2 Preferences	
Sort ○ Unsorted ○ By Item ○ By Name ○ By Package	Package COMMON-LISP- Show Package Names Maximum Items 500 Attribute Length 100 1
<u>0</u> K	Cancel Apply

Figure 13.6 Inspector Preferences dialog

Choose the sort option that you require from those listed in the Sort panel:

By Item	Sorts items alphabetically according to the printed representation of the item.
By Name	Sorts items alphabetically according to their names. This is the default setting.
By Package	Sorts items alphabetically according to the packages of the name field.
Unsorted	Leaves items unsorted. This displays them in the order they were originally defined.

In the Package panel, specify the name of the process package for the inspector. Select **Show Package Names** if you want package names to be displayed in the inspector. See Section 3.7 on page 29 for more details.

The Maximum panel contains options to let you configure the amount of information displayed in the inspector.

Choose a value from the Attribute Length drop-down list box to limit the length of any attributes displayed in the main list (that is, the contents of the first column in the list). The default value is 100 characters, and the minimum allowable value is 20 characters.

Choose a value from the Items drop-down list box to limit the number of items displayed in the main list. By default, 500 items are shown.

If you inspect an object that has more than the maximum number of items, then the excess items are grouped together in a list which itself becomes the last item displayed in the main list. Double-clicking on this inspects the remaining items for the object.

If necessary, the inspector splits any remaining items into several lists, all linked together in this fashion. For instance, if you limit the maximum number of items to 10, and inspect an object with 24 items, the inspector displays the first 10, together with an 11th entry, which is a list containing the next ten items. Double-clicking on this shows the next ten items, together with an 11th entry, which is a list containing the last four items. This is illustrated in Figure 13.7 below.



Figure 13.7 Displaying an object with more items than can be displayed

13.5 Customizing the inspector

The inspector provides two additional commands in the listener view.

Check **Bind \$ to the current inspector object** to bind the variable **\$** to the current object in the inspector in the listener.

Check **Automatically inspect listener values** to inspect listener values automatically.

Both options are checked by default.
13.6 Creating new inspection formats

There is a default inspection format for each Lisp object. If you want to customize the Common LispWorks inspector, you can change the inspection format of a class by defining new methods on get-inspector-values. See the *LispWorks Reference Manual* for a full description.

The Common LispWorks inspector tool can be customized by adding new inspection formats. To do this, you need to define new methods on the generic function get-inspector-values.

get-inspector-values takes two arguments: *object* and *mode*, and returns 5 values: *names*, *values*, *getter*, *setter* and *type*.

object	The object to be inspected.
mode	This argument should be either nil or eql to some other symbol. The default format for inspecting any object is its nil format. The nil format is defined for all Lisp objects, but it might not be sufficiently informative for your classes and it may be overridden.
names	The slot-names of <i>object</i> .
values	The values of the slots corresponding to <i>names</i> . The inspector displays the <i>names</i> and <i>values</i> in two columns in the scrollable pane.
getter	This is currently ignored. Use nil.
setter	This is a function that takes four arguments: an object (of the same class as <i>object</i>), a slot-name, an index (the position of the slot-name in <i>names</i> , counting from 0), and finally a new-value. (It is usual to ignore either the slot-name or the index.) This function should be able to change the value of the appropriate slot of the given object to the new-value.

type This is the message to be displayed in the message area of the inspector. This is typically either *mode* or—if *mode* is nil—then the name of the class of *object*.

13.6.1 Example

Consider the following implementation of doubly-linked lists.

```
(in-package "DLL")
(defstruct (dll (:constructor construct-dll)
                (:print-function print-dll))
 previous-cell
 value
 next-cell)
(defun make-dll (&rest list)
  (loop with first-cell
        for element in list
        for previous = nil then cell
        for cell = (construct-dll :previous-cell cell
                                   :value element)
        doing
        (if previous
            (setf (dll-next-cell previous) cell)
          (setq first-cell cell))
        finally
        (return first-cell)))
(defun print-dll (dll stream depth)
  (declare (ignore depth))
  (format stream "#<dll-cell ~A>" (dll-value dll)))
```

You can inspect a single cell by inspecting the following object:

(dll::make-dll "mary" "had" "a" "little" "lamb")

The resulting inspector has three slots: dll::previous-cell with value nil, value with value "mary" and dll::next-cell with value #<dll-cell had>.

In practice, you are more likely to want to inspect the whole doubly-linked list in one window. To do this, define the following method on get-inspectorvalues.

```
(in-package "DLL")
```

```
(defun dll-root (object)
  (loop for try = object then next
        for next = (dll-previous-cell try)
        while next
        finally
        (return try)))
(defun dll-cell (object number)
  (loop for count to number
        for cell = object then (dll-next-cell cell)
        finally
        (return cell)))
(defmethod system::get-inspector-values ((object dll)
  (mode (eql 'follow-links)))
  (let ((root (dll-root object)))
     (values
       (loop for cell = root then (dll-next-cell cell)
             for count from 0
             while cell
             collecting count)
       (loop for cell = root then (dll-next-cell cell)
     while cell
             collecting (dll-value cell))
       nil
       #'(lambda (object key index new-value)
     (declare (ignore key))
     (setf (dll-value (dll-cell (dll-root object) index)) new-
value))
       "FOLLOW-LINKS")))
```

Inspecting the same object with the new method now displays five slots, numbered from 0 to 4 with values "mary" "had" "a" "little" and "lamb".

The following example modifies the above call to get-inspector-values so that it inspects cells rather than their value slots. The setter updates the next-cell. Use this new mode to inspect the "lamb" cell and then set its next-cell slot to (make-dll "with" "mint" "sauce"). The expanded sentence can now be inspected in the follow-links mode.

(in-package "DLL")

```
(defmethod system::get-inspector-values
 ((object dll) (mode (eql 'follow-cells)))
  (let ((root (dll-root object)))
     (values
       (loop for cell = root then (dll-next-cell cell)
             for count from 0
             while cell
             collecting count)
       (loop for cell = root then (dll-next-cell cell)
     while cell
             collecting cell)
       nil
       #'(lambda (object key index new-value)
     (declare (ignore key))
    (setf (dll-next-cell (dll-cell (dll-root object) index)) new-
value))
       "FOLLOW-CELLS")))
```

Example: Using The Interface Builder

This example shows you how to use the interface builder to design a simple interface. It explains how to create the layout and the menu system, and demonstrates some of the attributes that you can set. Finally, the interface is saved to a file, and combined with some other simple code to produce a working example. You are strongly advised to read Chapter 15, "The Interface Builder", before (or in conjunction with) this chapter. It is also useful, though not essential, if you are familiar with the editor (Chapter 10), the listener (Chapter 16), and Common Lisp systems.

The final interface created is shown in Figure 14.1. It consists of a column layout which contains a graph pane, a display pane, and a list panel.



Figure 14.1 Example interface

Any select action performed in either the graph pane or the list panel is described in the display pane. This includes the following actions:

- Selecting any item
- Deselecting any item
- Extending the selection (by selecting more than one item)

Double-clicking any item in either the graph pane or the list panel displays a dialog which shows which item you double-clicked.

Lastly, there are menu commands available which display, in a dialog, the current selection in either pane. Choose **Selection > Graph** to see the currently selected items in the graph pane, and choose **Selection > List Panel** to see the currently selected items in the list panel.

14.1 Creating the basic layout

This section shows you how to create the basic layout for your interface, without specifying any attributes. Normally, this stage is literally the work of only a few seconds. The process is described in some detail here, to illustrate the way that the interface builder ensures that the most appropriate item is selected in the graphs of both the layouts and menus views, so as to minimize the steps you need to take when creating an interface.

- 1. Create an interface builder, if you do not already have one.
- 2. Choose File > New.

A new, empty, interface skeleton appears.

3. If the layouts view is not displayed, click the Layouts tab in the interface builder.

To begin, you need to add the main column layout to the interface using the buttons panels at the bottom of the interface builder. The Layouts tab at the bottom of the interface builder (as distinct from the Layouts tab you use to switch to the layouts view), lists the different types of layout that you can add to an interface.

4. Click Column in the button panel.

A column layout object is added as a child of the interface object. Nothing appears in the interface skeleton yet, since a column layout is a container for other window objects, and cannot itself be displayed. Note that the column layout remains selected in the layout graph. This is because column layouts are objects which can themselves have children, and the interface builder assumes that you are going to add some children next.

5. In the button panel, click the Graph Panes tab, and then click **Graph** to add a graph pane to the interface.

The graph pane object is added as the child of the column layout, and a graph pane appears in the interface skeleton.

- 6. Next, click the Basic Panes tab and then click Display.
- 7. Next, click List Panel.

The objects that you specify are added to the interface, and the interface skeleton is updated accordingly. Note that the column layout object remains selected throughout. You have now created the basic layout for the interface.

Next, suppose that you decide to add a title to the left of the display pane. You might want to do this make it clear what information is being shown in the display pane.

To do this, you can create a new row layout, add a title pane to it, and then move the existing display pane into this new row-layout. In addition, you must reorganize some of the elements in the interface.

1. Ensure that Column-Layout-1 is still selected in the Layout hierarchy area.

The new row layout needs to be added as a child of the column layout.

- **2.** In the button panel at the bottom of the interface builder, click the Layouts tab to display the available layouts once more.
- 3. Click on Row.

Notice that the new row layout remains selected, ready for you to add objects to it.

4. Click the Basic Panes tab again, and click Title.

Next, you must move the display pane you have already created, so that it is contained in the new row layout.

- 5. In the Layout hierarchy area, select Display-Pane-1 and choose Edit > Cut.
- 6. Select Row-Layout-1 and choose Edit > Paste.

The items have already been placed in the row layout in the positions you want them. However, the row layout itself has been added to the bottom of the interface; you want it to be in the same position as the display pane you initially created. To do this, move the list panel to the bottom of the interface.

7. Select List-Panel-1 and choose Object > Lower from the menu bar on the interface builder itself.

You have now finished creating the layout for the example interface. The next step is to name the elements of the interface in a sensible fashion.

14.2 Specifying attribute values

As you have already seen, the interface builder assigns default names such as Row-Layout-1 to the elements you add to an interface; you usually want to replace these with your own names. In addition, there are probably titles that you want to add to the interface; you can see the default titles that have been created by looking at the interface skeleton. The next stage of the example shows you how to change these default names and titles.

Changing the name or title of an element is actually just a case of changing the value assigned to an attribute of that element, as described in Section 15.5.4 on page 210. You would normally assign values to a number of different attributes at once, rather than concentrating on the names and titles of elements. The example is structured in this way to give you an idea of the sort of working practices you might find it useful to adopt when generating interface code.

To recap, the layout hierarchy of the example interface is shown in Figure 14.2. To ensure that you can understand this layout easily in the future, it is important to assign meaningful names and titles to the elements it contains now.



Figure 14.2 Layout hierarchy of the example interface

 Select the INTERFACE-1 object and press Return. The Attributes dialog appears as shown in Figure 14.3.

🛃 Interf	ace Builder 1				×
Attributes	for Interface-2:				
Basic	Advanced Title		Callbacks	Geometry	Style
	Best Height:				
	Best Width:				
	Best X:				
	Best Y:				
Co	lor Requirements:				
Confirm	Destroy Function:				
	Control Id:				
	Default Height:				
	Default Width:				
	Default X:				
	Default Y:				
	Enabled:				
	Message Area:				
	Name:	INTE	RFACE-1		
	Override Cursor:				
	Owner:				
	Pane Menu:				
	Title:	''Inte	rface-1"		_
			<u>0</u> k		ancel

Figure 14.3 Attributes dialog for the example interface

Notice that the NAME attribute of the interface has the value INTER-FACE-1, and the TITLE attribute has the value "Interface-1".

Note: If this is not the first interface you have created in the current session, the number is different.

- 2. Delete the value in the NAME text box, and type ib-example.
- 3. Delete the value in the TITLE text box, and type "Example Interface".
- 4. Click OK to dismiss the Attributes dialog and update the interface.

The name of the interface is now displayed as IB-EXAMPLE in the Layout hierarchy area, and the title of the interface skeleton changes to Example Interface.

Note: Case is not significant in the NAME attribute, because it is a Common Lisp symbol, but it is significant in the TITLE attribute, which is a string.

5. Select the Column-Layout-1 element, display its Attributes dialog, and change the value of its NAME attribute to main-layout.

Now change the names of the other objects in the interface.

- 6. Select the graph pane and change its NAME attribute to graph, and its INTERACTION attribute to :extended-selection. Click OK.
- 7. Select the list panel and change its NAME attribute to list, and its INTERACTION attribute to :extended-selection. Do not click **OK** yet.

The value of the INTERACTION attribute allows you to select several items from the list panel and the graph pane, using the appropriate method for your platform.

8. Change the ITEMS attribute of the list panel to the following string:

("One" "Two" "Three" "Four" "Five" "Six" "Seven" "Eight")

9. Click OK.

The row layout you created contains objects which are used solely to display information.

- **10.** Select the row layout object and change its NAME attribute to display-layout.
- 11. Change the ADJUST attribute of the row layout to :center. Click OK.

Changing the ADJUST attribute centers the title pane and the display pane vertically in the row layout, which ensures they line up more pleasingly.

In the working example, the display-layout object is going to show information about the current selection, so you need to change the names and titles of the objects it contains accordingly.

- 12. Select the title pane and change its NAME attribute to selection-title and its TEXT attribute to "selection:". Click OK.
- 13. Select the display pane and change its NAME attribute to selectiontext, and its TEXT attribute to "Displays current selection". Click OK.

This specifies a text string that is displayed when the interface is initially created. This string disappears as soon as you perform any action in the interface.

The layout hierarchy is now as shown in Figure 14.4. The names that you have assigned to the different objects in the interface make the purpose of each element more obvious.



Figure 14.4 Layout hierarchy with names specified

14.3 Creating the menu system

Next, you need to create a menu system for the example interface. This section shows you how to create the basic objects which comprise it.

1. Click the Menus tab in the interface builder to switch to the menus view.

A menu bar is created automatically when you create a new interface. To create the menu system for the example interface, you need to add a menu which contains two items.

- 2. Select the Menu-Bar object in the Menu hierarchy area.
- **3.** Click **Menu** to create the menu, then click **Item** twice to create the two items in the menu.

Notice that, as in the layouts view, an object remains selected if it can itself have children. This means that creating the basic menu structure is a very easy process.

Next, you need to name the objects you have created. As with the layouts, this is achieved by specifying attribute values.

- 4. Make sure that the Menu-1 menu is still selected, and press Return to display its Attributes dialog.
- 5. Change its NAME attribute to selection-menu. Do not click OK yet.

As well as specifying the NAME attribute for the menu you created, you need to change the TITLE attribute of each object you created. To do this, you must ensure that the appropriate attribute categories are displayed in the Attributes dialog.

6. Click on the Title tab in the Attributes dialog.

The Attributes dialog changes to appear as shown in Figure 14.5.



Figure 14.5 Displaying title attributes for a menu

7. In the Title tab view of the Attributes dialog, change the TITLE attribute to "selection". Click **OK**.

The TITLE attribute is used to specify the title of the menu that appears in the interface itself; note the change in the interface skeleton.

Next, you need to change the attributes of the two menu items.

- 8. Select the "Item-1" object and press Return.
- **9.** In the Attributes dialog, change the TITLE attribute to "Graph" and the NAME attribute to graph-command. Click **OK**.
- 10. Select the "Item-2" object, display its Attributes dialog and change the TITLE attribute to "List Panel" and the NAME attribute to list-panel-command. Click OK.

You have now finished the basic definition of the menu system for your example interface.

14.4 Specifying callbacks in the interface definition

The interface that you have designed contains a complete description of the layouts and menus that are available, but does not yet specify what any of the various elements do. To do this, you need to specify callbacks in the interface definition. As you might expect, this is done by setting attribute values for the appropriate elements in the interface.

In this example, the callbacks that you supply are calls to other functions, the definitions for which are assumed to be available in a separate source code file, and are discussed in Section 14.6. Note that you do not have to take this approach; you can just as easily specify callback functions within the interface definition itself, using lambda notation. It is up to you whether you do this within the interface builder, or by loading the code in the editor. If you choose the former, note that it may be easier to use the code view, rather than typing lambda functions into the Attributes dialog.

14.4.1 Specifying layout callbacks and other callback information

This section shows you how to specify all the callbacks necessary for each element in the example interface, together with other attributes that are required for correct operation of the callback functions. You need to specify attribute values for the display pane, the list panel and the graph pane.

1. If necessary, click the Layouts tab at the top of the interface builder to display the layouts view.

For the display pane, you only need to specify one attribute.

- 2. Select the selection-Text display pane and display the Attributes dialog.
- Set the READER attribute to selection-reader.
 This reader allows the display pane to be identified by the callback code.
 For the list panel, you need to specify four callbacks and a reader.
- 4. Select the List list panel and display its Attributes dialog.
- 5. Set the READER attribute to list-reader. Do not click **OK** yet.

Like the display pane, this reader is necessary so that the list panel can be identified by the callback code.

Next, you need to specify the following four types of callback:

Selection callback. The function that is called when you select a list item.

Extend callback. The function that is called when you extend the current selection.

Retract callback. The function that is called when you deselect a list item.

Action callback. The function that is called when you double-click on a list item.

Callbacks attributes are not currently visible in the Attributes dialog, so you must display them first.

- **6.** Click **Options**, and ensure that **Callbacks** is selected in the Attributes to View dialog and click **OK**.
- 7. Now set the following attributes of the list panel. Click **OK** when done.

SELECTION-CALLBACK to 'update-selection-select EXTEND-CALLBACK to 'update-selection-extend RETRACT-CALLBACK to 'update-selection-retract ACTION-CALLBACK to 'display-selection-in-dialog

8. Select the Graph graph pane and display its Attributes dialog.

For the graph pane, you need to set the same four callbacks, as well as a reader, and two other attributes that are important for the callback code to run correctly.

9. Set the following attributes of the graph pane.

SELECTION-CALLBACK to 'update-selection-select EXTEND-CALLBACK to 'update-selection-extend RETRACT-CALLBACK to 'update-selection-retract ACTION-CALLBACK to 'display-selection-in-dialog

- **10.** Set the READER attribute to graph-reader.
- 11. Set the CHILDREN-FUNCTION attribute to 'children-function.

The children function defines what is drawn in the graph, and so is vital for any graph pane.

12. Set the NODES-CONTRACTABLE-P attribute to nil.

Normally, double-clicking on a graph node expands or collapses the node, as appropriate. In this example, the action callback calls a different function explicitly. To ensure that this function is always called, you must inhibit the normal graph behavior by setting this attribute to nil. If you do not do this, then the action callback you have specified is only called on nodes which cannot be expanded or contracted (that is, leaf nodes).

13. Click OK to dismiss the Attributes dialog.

Note: The graph pane no longer displays a graph. This is because you have specified a children function which is not yet defined in the environment. You may also find that a notifier appears, informing you that the function is not defined. Click **Abort** to remove the notifier.

14.4.2 Specifying menu callbacks

The callbacks that are necessary for the menu system are much simpler than for the layouts; the example interface only contains two menu commands, and they only require one callback each.

1. Click the Menus tab to switch to the menus view.

- 2. Choose the "Graph" menu item, display its Attributes dialog and change the CALLBACK attribute to 'display-graph-selection. Click OK.
- 3. Choose the "List Panel" menu item, display its Attributes dialog and change the CALLBACK attribute to 'display-list-selection. Click OK.

14.5 Saving the interface

If you have followed this example from the beginning, the interface is now completely specified. You can now save the source code definition in a file.

Choose File > Save or click location to save the interface definition. Choose a directory in the dialog that appears, and specify the filename ib-example.lisp in the "File name" text box.

The file ib-example.lisp is displayed in the editor.

14.6 Defining the callbacks

This section shows you how to create the callback functions you need to define in order to complete the working example.

- 1. Choose File > New or click 🗋 to create a new file.
- Choose File > Save or click to save the file. Save it in the same directory you saved ib-example.lisp, and call this new file ib-call-backs.lisp.
- **3.** In the editor, specify the package for the callback definitions by typing the following into the ib-callbacks.lisp file:

(in-package "COMMON-LISP-USER")

- 4. Type the function definitions given in the rest of this section.
- **5.** Choose **File > Save** or click **I** to save the file when you have typed in all the function definitions.

The functions that you need to define in this file are divided into the following categories:

• Callbacks to update the display pane.

- Callbacks to display data in a dialog.
- Callbacks for menu items.
- Other miscellaneous functions.

14.6.1 Callbacks to update the display pane

One main function, update-selection, serves to update the display pane whenever selections are made in the graph pane or the list panel.

```
(defun update-selection (type data interface)
  (setf (capi:display-pane-text (selection-reader interface))
        (format nil "~A ~A" data type)))
```

The following three functions are the callbacks specified whenever a select, retract or extend action is performed in either the list panel or the graph pane. Each function is named according to the type of callback it is used for, and it simply calls update-selection with an additional argument denoting the callback type.

```
(defun update-selection-select (&rest args)
  (apply 'update-selection "selected" args))
(defun update-selection-retract (&rest args)
  (apply 'update-selection "deselected" args))
(defun update-selection-extend (&rest args)
  (apply 'update-selection "extended" args))
```

14.6.2 Callbacks to display data in a dialog

As with update-selection, one main function serves to display the data from any action in a dialog.

```
(defun display-in-dialog (type data interface)
 (capi:display-message
  "~S: ~A ~S"
   (capi:interface-title interface) type data))
```

The function display-selection-in-dialog is the action callback for both the graph pane and the list panel. It calls display-in-dialog, specifying one of the required arguments.

```
(defun display-selection-in-dialog (&rest args)
  (apply 'display-in-dialog "selected" args))
```

Note: Although only one action callback is specified in the example interface, the relevant functions have been defined in this modular way to allow for the possibility of extending the interface. For instance, you may decide at a later date that you want to display the information for an extended selection in a dialog, rather than in the display pane. You could do this by defining a new callback which calls display-in-dialog, passing it an appropriate argument.

14.6.3 Callbacks for menu items

Both menu items in the interface need a callback function. As with other callback functions, these are specified by defining a general callback, displaypane-selection, which displays, in a dialog, the current selection of any pane.

The following two functions call display-pane-selection, passing the reader of a pane as an argument. These functions are specified as the callbacks for the two menu items.

```
(defun display-graph-selection (&rest args)
  (apply 'display-pane-selection 'graph-reader args))
(defun display-list-selection (&rest args)
  (apply 'display-pane-selection 'list-reader args))
```

As with the other callback functions, specifying the callbacks in this way allows for easy extension of the example.

14.6.4 Other miscellaneous functions

Graph panes require a function which is used to plot information, called the children function. The value of the ROOTS attribute of a graph is passed as an

argument to the children function in order to start the plot. The example interface uses the following simple children function.

```
(defun children-function (x)
  (when (< x 8)
        (list (* x 2) (1+ (* x 2)))))</pre>
```

Note: The ROOTS attribute of a graph pane has a default value of *'*(1). This is generated automatically by the interface builder.

Finally, the function test-ib-example is used to create an instance of the example interface.

```
(defun test-ib-example ()
  (capi:display (make-instance 'ib-example)))
```

14.7 Creating a system

If you have followed this example from the beginning, the interface and its callbacks are now completely specified. Next, you can create a Common Lisp system which integrates the interface definition with the callback code.

- 1. Choose File > New or click . This creates a new, unnamed file in the editor.
- 2. Type the following form into this new file:

```
(defsystem ib-test
 (:package "CL-USER")
 :members ("ib-callbacks" "ib-example"))
```

This form creates a system called *ib-test* that contains two members; *ib-example.lisp* (the file containing the interface definition) and *ib-callbacks.lisp* (the file containing the callback code).

3. Choose File > Save or click 🖃 to save the new file. Save it in the same directory that you saved the ib-example.lisp and ib-callbacks.lisp files, and call this file defsys.lisp.

14.8 Testing the example interface

You have now finished specifying the example interface and its callback functions, so you can test it.

- Choose File > Save or click to save ib-example.lisp, ib-call-backs.lisp, and defsys.lisp if you have not already done so. Next, you need to load the ib-test system into the environment.
- In the editor, make sure that the file defsys.lisp is visible, and choose
 File > Compile and Load to compile and load it into the environment.
- 3. Press Space when the compilation has finished.
- **4.** In the listener, type the following form.

```
(load-system 'ib-test)
```

The ib-test system, together with its members, is loaded into the environment.

5. To test the interface, type the following form into the listener.

```
(cl-user::test-ib-example)
```

A fully functional instance of the example interface is created for you to experiment with.

The Interface Builder

The interface builder lets you construct graphical user interfaces (GUIs) for your applications. You design and test each window or dialog in your application, and the interface builder generates the necessary source code to create the windows you have designed; all you need to do is add callbacks to the generated code so that your own source code is utilized.

As you create each window, it is automatically displayed and updated onscreen, so that you can see what you are designing without having to type in, evaluate, or compile large sections of source code.

As well as making code development significantly faster, the interface builder allows you to try out different GUI designs, making it easier to ensure that the final design best suits your users' needs.

The interface builder has three views to let you design an interface.

- The *layouts view* is used to specify the elements in each window or dialog of an application.
- The *menus view* is used to create menus and menu items for each window of an application.
- The *code view* lets you examine the source code that is automatically generated as you create an interface.

Like the debugger tool, the interface builder also has its own menu bar, containing commands that let you work with a loaded interface, or any of its components.

To create an interface builder, choose **Tools > Interface Builder** from the LispWorks podium.

& Interface I	Tuilder 2					- D X
Warks File	Edit Yew	Interface	Object	Histop	a Rep	
Interface: INTE	RFACE-3					
Lepout: Mer	ue Code					
Interface-3						
Lepout: Bas	ic Panes G	iraph Pane	Editor	Panes	Range Panes Bultons Pinboard Objects Interfaces Dividers	
Çolunn	Pinboard		Simple		Svitchable	
<u>G</u> éd	Вом	Sie	ngle Pinbo	bed	Tab	
0				_		

Figure 15.1 The interface builder

Because the interface builder generates source code which uses the CAPI library, this chapter assumes at least a minimum knowledge of the CAPI. See the *CAPI User Guide* and the *CAPI Reference Manual* for details.

A complete example showing you how to use the interface builder to design an interface, and how to integrate the design with your own code, is given in Chapter 14, "Example: Using The Interface Builder". You are strongly advised to work through this example after reading this chapter, or in conjunction with it.

15.1 Creating or loading interfaces

In the context of this chapter, an interface refers to any single window which is used in an application. Thus, an editor, an Open File dialog, or a confirmer containing an error message are all examples of interfaces. The GUI for a complete application is liable to comprise many interfaces. You can load as many different interfaces into the interface builder as you like, although you can only work on one interface at once. More formally, the class capi:interface is the superclass of all CAPI interface classes, which is the set of classes used to create elements for on-screen display. You can load any code which defines instances of this class and its subclasses into the interface builder.

Once you have invoked the interface builder, you can create new interfaces, or load any that have already been saved in a previous session. You must load or create at least one interface before you can proceed.

15.1.1 Creating a new interface

When you first start the interface builder, a new interface is created for you automatically. You can also choose **File > New** or click on to create a new interface. A blank Common LispWorks window, known as the *interface skeleton*, appears on-screen, as shown in Figure 15.2. The interface skeleton contains no layouts or panes, or menus.



Figure 15.2 Skeleton Common LispWorks window

You can use File > New to create as many interfaces as you want; they are all displayed as soon as you create them. Since you can only work on one interface at a time, use the History > Items submenu to switch between different interfaces that are currently loaded in the interface builder.

As an alternative, type the name of an interface directly into the Interface text box and press Return to create a new interface, or to switch to an interface which is already loaded.

15.1.2 Loading existing interfaces

Choose File > Open or click i to load an existing interface into the interface builder. You can load any CAPI interface, whether it is one that you have designed using the interface builder, or one that has been coded by hand using the CAPI library. You can load as many interfaces as you want, and then use the History > Items submenu to swap between the loaded interfaces when working on them.

To load one or more existing interfaces:

1. Choose File > Open.

A file prompter dialog appears.

2. Choose a file of Common Lisp source code.

You should choose a file that contains the source code for at least one CAPI interface. If the file does not contain any such definitions, an error message appears informing you of this.

Once you have chosen a file, a dialog appears listing all the interface definitions that have been found in the file, as shown in Figure 15.3. This lets you choose which interface definitions to load into the interface builder. By default, all the definitions are selected.

🛃 Interface Builder 2	×
Load interfaces from file:	
Button-Test There Button Evennle	<u>0</u> K
Image-Button-Example-2 Image-Button-Example-2	All
	<u>N</u> one
	<u>C</u> ancel

Figure 15.3 Choosing which interfaces to load into the interface builder

3. Select the interfaces you want to load into the interface builder and click **OK**.

You can select as many or as few of the listed interfaces as you like; the AII or **None** buttons can help to speed your selection. Click **Cancel** to cancel loading the interfaces altogether.

15.2 Creating an interface layout

The default view in the interface builder is the layouts view, as shown in Figure 15.4. You use this view to specify the entire GUI design, with the exception of the menus. Click the Layouts tab to swap to this view from any other in the interface builder.

Interface	Tuilder 2					- 0
arks Ele	Edit Yiew Inte	rlace Object History	Help			271,7000
face MAG	ACCE-MOTTUR-3	Abre-5				
eout: Mer	we Code					
age-Buto	n-Example-2	-Main Tw	ve Button Button ve Button sabler	81 62 63 64		
L.,	in Paner Grank	Parsen Folina Paran	Ranas Paner	Padrone Pintoneo	Objects Interface) Disistens
arout: Rw	THE R. LEWIS CO., LANSING MICH.	Contract Property in the little		1	a contract to the state	al room 1
Daharan	Britond	State	Doctrativ			

Figure 15.4 Displaying the layouts in the interface builder

The interface builder has three sections in the layouts view.

15.2.1 Interface box

The interface text box displays the name of the current interface; the interface that you are currently working on. Note that there may be several other interfaces loaded into the interface builder, but only one can be current.

To switch to another loaded interface, or to create a new interface, type the name of the interface into this area and press Return. The interface you specify appears and its layouts are shown in the interface builder.

15.2.2 Graph area

This area displays, in graph form, the CAPI elements of the current interface.

A popup menu gives you access to the standard graph commands described in Chapter 5, "Manipulating Graphs".

By default, the graph is laid out from left to right. The main interface name is shown at the extreme left, and the layouts and elements defined for that interface are shown to the right. The hierarchy of the layouts (that is, which elements are contained in which layouts, and so on) is immediately apparent in the graph.

An item selected in the graph can be operated on by commands in the **Object** menu in the interface builder's menu bar. This menu contains the standard action commands described in Section 3.8 on page 31, as well as a number of other commands described throughout this chapter.

15.2.3 Button panels

At the bottom of the interface builder are several tab layouts, each of which contains a large selection of buttons. These tabs list the types of CAPI interface class that can be used in the design of the current interface.

• Click the Layouts tab to see the different types of layout that you can use in an interface. This is the default tab and is displayed when you first switch to the layouts view.All other elements must be contained in layouts in order for them to be displayed.

- There are four different types of Panes tab: Basic, Graph, Editor and Range. Click on each to see the different types of pane that you can use in an interface.
- Click the Buttons tab to see the different types of button that you can use in an interface.
- Click the Pinboard Objects tab to see the different types of pinboard object that you can use in an interface.
- Click the Interfaces tab to see a number of types of pre-defined interface objects that you can use in an interface. These are interfaces which are already used throughout the Common LispWorks environment, and which may be useful in your own applications. Many of the tools described in this manual area listed in this pane.
- Click on the Dividers tab to select between Column and Row layout dividers.

15.2.4 Adding new elements to the layout

To add a new element to the layout, click the relevant button in any of the tabs in the button panel. The element is added as the child of the currently selected graph node. If nothing is currently selected, the element is added as the child of the *last* selected node.

Because construction of the interface layout is performed by selecting CAPI elements directly, you must be familiar with the way that these elements are used in the construction of an interface.

For instance, the first element to add to an interface is likely to be a CAPI layout element, such as an instance of the row-layout class or column-layout class. Not surprisingly, these types of element can be found in the Layouts tab of the button panel. Elements such as buttons or panes (or other layouts) are then added to this layout. In order to generate CAPI interfaces, it is important to know that all window elements must be arranged inside a layout element in this way.

When you add an element to the design, two windows are updated:

• The graph in the layout view is updated to reflect the position of the new element in the hierarchy.

• The interface skeleton is updated; the element that has been added appears.

When you add an item, an instance of that class is created. By default, the values of certain attributes are set so that the element can be displayed and the hierarchy layout updated in a sensible way. This typically means that name and title attributes are initialized with the name of the element that has been added, together with a numeric suffix. For instance, the first output pane that is added to an interface is called Output-Pane-1. You should normally change these attribute values to something more sensible, as well as set the values of other attributes. See Section 15.5 for details about this.

For a practical introduction to the process of creating an interface using the interface builder, see Chapter 14, "Example: Using The Interface Builder".

15.3 Creating a menu system

The menus view of the interface builder can be used to define a menu system for the current interface. Click the Menus tab to switch to the menus view from any other view in the interface builder. The interface builder appears as shown in Figure 15.5.

Interface Builder 2	
aks Elle Edit Yew Interface Object History Help	
face MAGE-BUTTON-EXAMPLE-2	
pouts Menus Code	
age-Buton-Example-2Menu-Bar	-
Herrafier Menu terr	

Figure 15.5 Displaying the menu structure of an interface

The menus view has two areas, together with six buttons which are used to create different menu elements. As with layouts, it is important to understand how CAPI menus are constructed. See the *CAPI User Guide* for details.

15.3.1 Interface box

This box is identical to the Interface box in the layouts view. See Section 15.2.1 for details.

15.3.2 Graph area

The graph area in the menus view is similar to the graph area in the layouts view. It displays, in graph form, the menu system that has been defined for the current interface. Menu items are displayed as the children of menus or menu components, which in turn are displayed as the children of other menus, or of the entire menu bar.

Like the layouts view, a new menu element is added as the child of the currently selected item in the graph, or the last selected element if nothing is currently selected.

15.3.3 Adding menu bars

A single menu bar is created in any new interface by default. This appears in the graph area as a child of the entire interface.

If you decide to delete the menu bar for any reason, use the **Menu Bar** button to create a new one.

15.3.4 Adding menus

To add a menu, click **Menu** on the button bar at the bottom of the interface builder. Each menu must be added as the child of the menu bar, or as the child of another menu or menu component. In the first case, the new menu is visible on the main menu bar of the interface. Otherwise, it appears as a submenu of the relevant menu.

Newly created menus cannot be selected in the interface skeleton until menu items or components are added to them.

By default, new menus are called MENU-1, MENU-2 and so on, and appear in the interface skeleton as Menu-1, Menu -2 and so on, as relevant. See Section 15.5 for details on how to change these default names.

15.3.5 Adding menu items

To add a menu item to the current interface, click **Item** on the button bar. Each menu item must be added as the child of either a menu or a menu component.

If added as the child of a menu component, new items have a type appropriate to that component; see Section 15.3.6 for details.

By default, new menu items are named ITEM-1, ITEM-2, and so on, and are displayed in the interface skeleton as Item-1, Item-2 and so on, as relevant. See Section 15.5 for details on how to change these default names.

15.3.6 Adding menu components

Menu components are an intermediate layer in the menu hierarchy between menus and menu items, and are used to organize groups of related menu items, so as to provide a better structure in a menu system.

There are three types of menu component which can be defined using the classes in the CAPI interface library:

- Standard menu components.
- Radio components.
- Check components.

15.3.6.1 Standard menu components

A standard menu component can be used to group related menu commands that would otherwise be placed as direct children of the menu bar they populate. This offers several advantages.

- Related menu items (such as **Cut**, **Copy**, and **Paste**) are grouped with respect to their code definitions, as well as their physical location in an interface. This encourages a more logical structure which makes for a better design philosophy.
- Using standard menu components to group related items is particularly useful when re-arranging a menu system. Groups of items may be moved in one action, rather than moving each item individually.
- Grouping items together using standard menu components adds a separator which improves the physical appearance of any menu.

Click **Component** in the button bar to add a standard menu component to the current interface. Menu components must be added as the children of a menu.

Menu components are not visible in the interface skeleton until at least one item or submenu has been added, using the **Item** or **Menu** buttons.

Menu items added to a standard menu component appear as standard menu items in that component.

15.3.6.2 Radio components

A radio component is a special type of menu component, in which one, and only one, menu item is active at any time. For any radio component, one item always returns t, and the others always return nil. The menu item that was selected last is the one that returns t.

Radio components are used to group together items, only one of which may be chosen at a time.

Click **Radio Component** in the button bar of the interface builder to add a radio component to the current interface. Radio components must be added as the children of a menu, and, like standard menu components, are not visible in the interface skeleton until items have been added. To add an item to a radio component, click **Item**. New items are automatically of the correct type for radio components. Note that you cannot add a submenu as an item in a radio component.

The way that radio components are indicated on-screen depends on the operating system or window manager you are running; a dot to the left of the selected item is common. On some systems, a diamond button is placed to the left of every item, and this is depressed for the item which is currently selected.

Like standard menu components, separators divide radio components from other items or components in a given menu.

15.3.6.3 Check components

Like radio components, check components place constraints on the behavior of their child items when selected. Each item in a check component either returns t or nil, and repeatedly selecting a given item toggles the value that is returned. Thus, check components allow you to define groups of menu items which can be turned on and off independently.

An example of a check component in the Common LispWorks environment are the commands in the **Tools > Customize** menu, available from any window in the environment.

Click **Check Component** in the button bar of the interface builder to add a check component to the current interface. Like other components, check components must be added as the children of a menu, and are not visible until items have been added. Use the **Item** button to add an item to a check component; it is automatically given the correct menu type. Note that you cannot add a submenu as an item in a radio component.

Like radio components, the way that check components are indicated onscreen depend on the window manager or operating system being used. A tick to the left of any items which are "switched on" is typical. Alternatively, a square button to the left of check component items (depressed for items which are on) may be used.

15.4 Editing and saving code

As you create an interface in the interface builder, source code for the interface is generated. You can use the code view to examine and, if you want, edit this code. You can also save the source code to disk for use in your application. This section discusses how to edit and save the code generated by the interface builder, and discusses techniques which let you use the interface builder in the most effective way.

15.4.1 Integrating the design with your own code

As the design for a GUI evolves, it is useful to include the code generated by the interface builder with your own code, in order to produce a working application that you can test.

The flexible design of the interface builder means that you can use the software development strategy that you find the most appropriate. You do not have to change the way you work in order to get the best out of the interface builder. Rather, your usage of the interface builder fits in with your own working practice.

At one extreme, you can specify the entire GUI for an application using the interface builder: even callbacks, keyboard accelerators for menu items, and
so on. This means that the source code for the entire GUI is generated automatically, and can then be integrated with your own hand-written source code in the manner which suits you best, perhaps by using Common Lisp systems, as described in the *LispWorks User Guide*.

Alternatively, you may prefer to use the interface builder for the basic design only. Once you have created an interface skeleton that you are happy with, you could start to augment the automatically generated source code with hand-written code. At this stage, you could dispense with the interface builder completely if you wished.

There are, of course, many stages between these two extremes. Because of the iterative nature of software evolution, the interface builder is often best used in parallel with other tools. A cyclic work pattern is often most effective, whereby you develop part of the interface (using the interface builder) together with the underlying code (using other tools), then save and test the results, and then go back and fine-tune the code you have already developed.

The interface builder is flexible enough that you can use it to whatever level you feel is most suited to your working methods, and to the needs of the application itself.

15.4.2 Editing code

Click the Code tab to switch to the code view. You can use this view to display and edit the code that is generated by the interface builder. The interface builder appears as shown in Figure 15.6.

```
🛃 Interface Builder 2
                                                                                             _ 🗆 ×
Works File Edit View Interface Object History Help
Interface: IMAGE-BUTTON-EXAMPLE-2
Layouts Menus Code
                                                                                            <u>U</u>pdate
Capi:define-interface image-button-example-2 ()
   n.
   (:panes
    (buttons1
     capi:radio-button-panel
     :items '(1 2)
     :images (list :do :do)
    :disabled-images (list :do-disabled :do-disabled)
     :selected-images (list :dont :dont)
     :selected-disabled-images (list :dont-disabled :dont-disabled)
     :indicator t)
    (buttons2
     capi:radio-button-panel
     :items '(1 2)
     :images (list :do :do)
     :disabled-images (list :do-disabled :do-disabled)
     :selected-images (list :dont :dont)
     :selected-disabled-images (list :dont-disabled :dont-disabled)
     :indicator nil)
    (buttons3
     capi:check-button-panel
     :items '(1 2)
     :images (list :do :do)
     :disabled-images (list :do-disabled :do-disabled)
     :selected-images (list :dont :dont)
     :selected-disabled-images (list :dont-disabled :dont-disabled)
     :indicator t)
    (buttons4
     capi:check-button-panel
     :items '(1 2)
     :images (list :do :do)
     :disabled-images (list :do-disabled :do-disabled)
     :selected-images (list :dont :dont)
```

Figure 15.6 Displaying source code in the interface builder

Like the other views in the interface builder, an Interface box at the top of the code view displays the name of the current interface. See Section 15.2.1 for details.

The rest of this view is dedicated to a an editor window that displays the code generated for the interface. Like other editor windows in Common LispWorks, all the keyboard commands available in the built-in editor are available in the Code area. If you want to change the definition of the current interface by editing the source code directly, edit the code in the Code area and then click **Update**. Any changes you have made are reflected in the interface skeleton.

15.4.3 Saving code

There are several ways to save the code generated by the interface builder into files of source code. Any files that you save are also displayed as buffers in the editor.

Choose File > Save or click it to save the current interface. If it has already been saved to a file, the new version is saved to the same file. If the interface has not been saved before, you are prompted for a filename. After saving, the file is displayed in the editor.

Choose File > Save As to save the current interface to a specific file. This command always prompts you for a filename; if the interface has not been saved before, this command is identical to File > Save, and if the interface has already been saved, this command saves a copy into the file you specify, regardless of the file it was originally saved in. After saving, the file is displayed in the editor.

Choose File > Save All to save all of the interfaces that have been modified. A dialog allows you to specify precisely which interfaces to save. Choosing this command is analogous to choosing File > Save individually for each of the interfaces you want to save. If there are any interfaces which have not been saved previously, you are prompted for filenames for each one.

Choose File > Revert to revert the current interface to the last version saved.

Choose File > Close to close the current interface. You are prompted to save any changes if you have not already done so. The interface name is removed from the History > Items submenu.

Individual interface definitions are saved in an intelligent fashion. You can specify the same filename for any number of interfaces without fear of overwriting existing data. Interface definitions which have not already been saved in a given file are added to the end of that file, and existing interface definitions are replaced by their new versions. Source code which does not relate directly to the definition of an interface is ignored. In this way, you can safely combine the definitions for several interfaces in one file, together with other source code which might be unrelated to the user interface for your application.

Conversely, when loading interfaces into the environment (using File > Open or), you do not have to specify filenames which only contain definitions of interfaces. The interface builder scans a given file for interface definitions, loads the definitions that you request, and ignores any other code that is in the file. See Section 15.1.2 for details on loading interfaces into the interface builder.

This approach to saving and loading interface definitions ensures that your working practices are not restricted in any way when you use the interface builder to design a GUI. You have complete control over the management of your source files, and are free to place the source code definitions for different parts of the GUI wherever you want; the interface builder can load and save to the files of your choice without failing to load interface definitions and without overwriting parts of the source code which do not relate directly to the GUI.

15.5 Performing operations on objects

There are a large number of operations you can perform on any object selected in the graph of either the layouts view or the menus view. These operations allow you to refine the design of the current interface.

The techniques described in this section apply to an object selected in either the layouts view or the menus view. Any changes made are automatically reflected in both the interface builder and the interface skeleton.

15.5.1 Editing the selected object

As in any other Common LispWorkstool, you can use the commands in the **Edit** menu to edit the object currently selected in any graph of the interface builder. See Section 3.4 on page 24 for full details on the commands available.

15.5.2 Browsing the selected object

As in other tools, you can transfer any object selected in the graph into a number of different browsers for further examination. The standard action commands that let you do this are available in the **Object** menu. See Section 3.8 on page 31 for details.

15.5.3 Rearranging components in an interface

Rearranging the components of an interface in the most appropriate way is an important part of interface design. This might involve rearranging the layouts and window elements in an interface, or it might involve rearranging the menu system.

The main way to rearrange the components of an interface (either the layouts or the menu components) is to use the cut, copy and paste functions available, as described below.

To move any object (together with its children, if there are any):

- 1. Select the object in a graph in the interface builder (either the layouts view or the menus view, depending on the type of objects you are rearranging).
- 2. Choose Edit > Cut.

The selected object, and any children, are transferred to the clipboard. The objects are removed from the graph in the interface builder, and the interface skeleton.

- **3.** Select the object that you want to be the parent of the object you just cut. You must make sure you select an appropriate object. For instance, in the layouts view you must make sure you do not select a window element such as a button panel or output window, since window elements cannot have children. Instead, you should probably select a layout.
- 4. Choose Edit > Paste.

The objects that you transferred to the clipboard are pasted back into the interface design as the children of the newly selected object. The change is immediately visible in both the graph and the interface skeleton.

Note: You can copy whole areas of the design, rather than moving them, by selecting **Edit > Copy** instead of **Edit > Paste**. This is useful if you have a number of similar areas in your design.

The menu commands **Object > Raise** and **Object > Lower** can be used to raise or lower the position of an element in the interface. This effects the position of the element in the interface skeleton, the layout or menu hierarchy, and the source code definition of the interface. Note that these commands are available from the menu bar in the interface builder, rather than from the LispWorks podium.

15.5.4 Setting the attributes for the selected object

Choose **Object > Attributes** from the interface builder's menu bar to display the Attributes dialog for the selected object. This is shown in Figure 15.7. You can also double-click on an object to display this dialog.

The Attributes dialog lets you set any of the attributes available to the selected object, such as symbol names, titles, and callbacks. This gives you a high degree of control over the appearance of any object in the interface.

🛃 Interface Builder 2 👘	×
Attributes for Buttons4:	
Basic Advanced Title	Callbacks Geometry Style
Armed Images:	▲
Button Class:	
Cancel Button:	
Color Requirements:	
Control Id:	
Data Function:	
Default Button:	
Disabled Images:	0-DISABLED :D0-DISABLED)
Enabled:	
Images:	(LIST :DO :DO)
Indicator:	NIL
Items:	(1 2)
Key Function:	
Layout Args:	
Layout Class:	
Name:	BUTTONS4
Pane Menu:	
Print Function:	
	<u> </u>

Figure 15.7 Setting the attributes of the selected object

The precise list of attributes displayed in the dialog depends on the class of the object that you selected in the graph of the interface builder.

To set an attribute, type its value into the appropriate text box in the Attributes dialog. Click **OK** to dismiss the Attributes dialog when you have finished setting attribute values.

Because of the large number of attributes which can be set for any class of object, you can configure which ones are displayed in the Attributes dialog. For any object, the available attributes are divided into six general categories, as follows:

• Basic attributes

- Advanced attributes
- Title attributes
- Callbacks attributes
- Geometry attributes
- Style attributes

15.5.4.1 Basic attributes

These are the attributes that you are most likely to want to specify new values for. This includes the following information, depending on the class of the selected object:

- The name of the object.
- The items available (for list panels).
- The orientation and borders (for layouts).
- The text representation (for menu items).

15.5.4.2 Advanced attributes

This category lets you specify more advanced attributes of the selected object, such as its property list.

15.5.4.3 Title attributes

This category lets you specify the title attributes of the selected object. These attributes affect the way an object is titled on-screen.

15.5.4.4 Callbacks attributes

This category lets you specify any of the callback types available for the selected object. Many objects do not require any callbacks, and many require several.

15.5.4.5 Geometry attributes

This category lets you control the geometry of the selected object, by specifying any of the available height and width attributes. Geometry attributes are not available for menu objects.

15.5.4.6 Style attributes

This category lets you specify advanced style settings for the selected object. This includes the following attributes:

- The font used to display items in a list.
- The background and foreground colors of an object.
- The mnemonic used for a menu item.

15.6 Performing operations on the current interface

You can perform a number of operations on the current interface, using the commands in the **Interface** menu in the interface builder.

15.6.1 Setting attributes for the current interface

Choose Interface > Attributes to set any of the attributes for the current interface. An Attributes dialog similar to that shown in Figure 15.7 appears. You set attributes for the current interface in exactly the same way as you do for any selected object in the interface. See Section 15.5.4 for details.

15.6.2 Displaying the current interface

As already mentioned, an interface skeleton is automatically displayed when you load an interface into the interface builder, and any changes you make to the design are immediately reflected in the skeleton. There are also a number of commands which give you more control over the way that the interface appears on-screen as you work on its design.

Choose Interface > Raise to bring the interface skeleton to the front of the display. This command is very useful if you have a large number of windows onscreen, and want to locate the interface skeleton quickly. Choose Interface > Regenerate to force a new interface skeleton to be created. The existing interface skeleton is removed from the screen and a new one appears. This command is useful if you have changed the size of the window, and want to see what the default size is; this is especially applicable if you have altered the geometry of any part of the interface while specifying attribute values.

Regenerating the interface is also useful if you set an interface attribute which does not cause the interface skeleton to be updated automatically. This can happen, for instance, if you change the default layout of the interface, which you might want to specify if an interface has several views.

Many interfaces in a GUI are used in the final application as dialogs or confirmers. For such interfaces, the interface skeleton is not necessarily be the most accurate method of display. Choose Interface > Dialog or Interface > Confirmer to display the current interface as a dialog or as a confirmer, as appropriate. Dialogs are displayed without a menu bar, and with minimal window decoration, so that the window cannot be resized. Confirmers are similar to dialogs, but have OK and Cancel buttons added to the bottom of the interface. To remove a dialog, click in its Close box.

15.6.3 Arranging objects in a pinboard layout

Most types of layout automatically place their children, so that you do not have to be concerned about the precise arrangement of different objects in an interface. Pinboard layouts, however, allow you to place objects anywhere within the layout.

Objects which are added to a pinboard layout using the interface builder have borders drawn around them in the interface skeleton. You can interactively resize and place such objects by selecting and dragging these borders with the mouse.

When you have rearranged the objects in a pinboard layout to your satisfaction, choose **Interface > Display Borders**. This turns off the border display, allowing you to see the appearance of the final interface.

Note: You can only move and resize objects in a pinboard layout when borders are displayed in the interface skeleton. Choosing **Interface > Display Borders** toggles the border display.

15.6.4 Loading the current interface into the environment

Choose Interface > Evaluate to evaluate the definition for the current interface in the environment. Choose File > Compile from the LispWorks podium to compile the definition for the current interface in the environment. Using either of these commands allows you to create instances of the interface, either from the listener or from other code loaded into the environment. This is therefore a useful way of testing out your GUI design with your application source code.

If you have loaded a compiled version of your application source code into the environment, you should use File > Compile to load the current interface. If you have evaluated your application source code in the environment, then load the current interface with Interface > Evaluate. The Interface Builder

16

The Listener

The listener is a tool that lets you evaluate Common Lisp expressions interactively and immediately see the results. It is useful for executing short pieces of Common Lisp, and extensive use is made of it in the examples given in this manual. This chapter describes all the facilities of the listener.

16.1 The basic features of a listener

A listener is created automatically when you start Common LispWorks. You can also create a listener yourself by choosing **Tools > Listener** from the menu bar on the LispWorks podium.



Figure 16.1 Listener

In the listener view, the main area of the listener contains a prompt in its top left-hand corner.

Rather like the other types of command line prompt, this prompt helps you identify the point in the listener at which anything you type is evaluated. It

may also contain other useful information, such as the current package or the current number in the history list. Throughout this chapter, the prompt is shown in pieces of example code as *PROMPT* >.

You can click the Output tab to display the output view of the listener; this view displays any output that is created by the listener, or any child processes created from the listener.

As with other tools, commands available in the **Works** menu of the LispWorks podium are specific to the listener, when it is the active window.

To familiarize yourself with the listener, follow the instructions in the rest of this chapter, which forms a short lesson. Note that, depending on the nature of the image you are using, and the configuration that the image has been saved with, the messages displayed by Lisp may be different to those shown here.

16.2 Evaluating simple forms

1. Type the number 12 at the prompt, and press Return.

In general, assume that you should press Return after typing something at the prompt, and that you should type at the *current prompt* (that is, the one at the bottom of the screen). In fact, the latter is not always necessary; Section 16.9 on page 224 describes how to move the cursor to different places, and thus you may not always be on the current prompt.

Any Common Lisp form typed at the prompt is evaluated and its results printed immediately below in the listener.

Common Lisp evaluates this input and prints the result of that evaluation. When Common Lisp evaluates a number, the result is the number itself, and so 12 is printed out:

```
PROMPT > 12
12
PROMPT >
```

When results are printed in the listener, they start on the line following the last line of input. The 12 has been printed immediately below the first prompt, and below that, another prompt has been printed.

2. Type ***** at the current prompt.

```
PROMPT > *
12
PROMPT >
```

* always has as its value the result of the previous expression; in this case, 12, which was the result of the expression typed at the first prompt. For a full description, see the ANSI standard for Common Lisp (ANSI X3.226:1994), which is also available in this distribution.

3. Type (setg val 12) at the current prompt.

```
PROMPT > (setq val 12)
12
```

PROMPT >

The expression sets the variable val to 12. The result of evaluating the form is the value to which val has been set, and thus the listener prints 12 below the form typed at the prompt.

This is exactly the same behavior as before, when you typed a number it was evaluated and the result printed in the listener. What is different this time, of course, is that Lisp has been told to "remember" that 12 is associated with val.

4. Type val.

The form is evaluated and 12 is printed below it.

5. Type (+ val val val).

The form, which computes the sum of three vals, is evaluated, and 36 is printed below it.

16.3 Re-evaluating forms

If you change val to some other number, and want to know the sum of three vals again, you can avoid re-typing the form which computes it. To see how this is done, follow the instructions below.

1. Type (setq val 1).

The variable val is now set to 1.

2. Press alt+p.

PROMPT > (setq val 1)

The form you previously typed appears at the prompt. At this point, you could edit this form and press Return to evaluate the edited form. For the moment, just carry on with the next instruction.

3. Press Alt+P again, and then press Return.

```
PROMPT > (+ val val val)
3
PROMPT >
```

Pressing Alt+P a second time displayed the second to last form that you evaluated. This time, pressing Return immediately afterwards simply re-evaluates the form. Note that you could have edited the recalled form before evaluating it. You can use Alt+P repeatedly, recalling any form that you have evaluated in the current session.

This time the form evaluates to the number 3, because val changed in the interim.

16.4 Interrupting evaluation

The key sequence Ctrl+Break interrupts evaluation of interpreted Lisp forms. This is useful for stopping execution in the middle of a loop, or for debugging. When the interrupt is processed, the debugger is entered, with a continue restart available.

16.5 The History menu

The forms and commands typed at previous prompts are stored in the *history list* of the listener. It is so named because it records all the forms and commands you have typed into the listener. Many other command line systems have a similar concept of a history. Each form or command in the history is known as an *event*.

You can obtain a list of up to the last ten events in the history by displaying the **History > Items** submenu from the LispWorks podium. To re-evaluate any event, choose it from this submenu.

For more information about history lists in the environment, see Section 3.5 on page 26.

16.6 The Expression submenu

The **Works > Expression** menu lets you perform operations on the current expression, that is, the symbol in which the cursor currently lies. It behaves in exactly the same way as the **Works > Expression** menu in the editor.

Choose **Works > Expression > Documentation** to display the Common Lisp documentation for the current expression, if any exists. It is printed in a help window.

Choose **Works > Expression > Find Source** to search for the source code definition of the current expression. If it is found, the file is displayed in the editor: the cursor is placed at the start of the definition. See Chapter 10, "The Editor" for an introduction to the editor.

You can find only the definitions of expressions you have defined yourself (those for which you have written source code) — not those provided by the environment.

Choose **Works > Expression > Class** to look at the class of the current expression in a class browser. See Chapter 7, "The Class Browser" for full details about this tool.

Choose Works > Expression > Arguments to print the lambda list of the current expression in the echo area, if it is a function, generic function or method. This is the same as using the keyboard command Alt+=, except that the current expression is automatically used.

Choose **Works > Expression > Value** to display the value of the current expression in the echo area.

Choose **Works > Expression > Trace** to display a menu of trace commands which can be applied to the current expression. See Section 14.4 on page 215 for full details.

Choose **Works > Expression > Evaluate Region** to evaluate the Lisp code in the current region. You must make sure you have marked a region before choosing this command; see Section 10.11.1 on page 123. Whether you use the mouse or keyboard commands to mark a region does not matter.

Choose **Works > Expression > Compile Region** to compile the Lisp code in the current region.

Choose **Works > Expression > Macroexpand** to macroexpand the current form. The macroexpansion is printed in the output view, which is displayed automatically. Click the Output tab to redisplay the output at any time.

Choose **Works > Expression > Walk** to walk the current form. This performs a recursive macroexpansion on the form. The macroexpansion is printed in the output view, which is displayed automatically. Click the Output tab to redisplay the output at any time.

Choose **Works > Expression > Generic Function** to browse the current expression in a generic function browser. This command is only available if the current expression is a generic function. See Chapter 12, "The Generic Function Browser" for more details.

16.7 The Values submenu

The **Works > Values** menu lets you perform operations on the results of the last expression typed. The values returned from this expression are referred to as the *current values*.

The menu is not available if the most recently-typed expression was not a Common Lisp form. This is because the evaluation of the last expression typed must have produced at least one value to work on.

The **Works > Values** menu gives you access to the standard action commands described in Section 3.8 on page 31.

16.8 The Debug menu

This menu allows you to perform command line debugger operations upon the current stack frame. The menu is only available when the debugger has been invoked by some activity within the listener.

Some of the most commonly-used command line debugger commands are available from the **Debug** menu. You can also invoke the debugger tool from this menu.

Choose **Debug > Restarts** to display a submenu containing all the possible restarts for the debugger, including the abort and continue restarts. Choose any of the commands on this submenu to invoke the appropriate restart.

Choose Debug > Listener > Backtrace to produce a backtrace of the error.

Choose **Debug > Listener > Bindings** to display information about the current stack frame.

Choose **Debug > Condition > Find Source** to find the source code definition of the function at the current call frame and display it in an editor.

Choose **Debug > Listener > Next** to move to the next call frame in the stack.

Choose **Debug > Listener > Previous** to move to the previous call frame in the stack.

Choose **Debug > Listener > Start GUI Debugger** to invoke a debugger tool on the current error. See Chapter 9, "The Debugger Tool", for full details about using this tool.

Choose **Debug > Report Bug** to report a bug in the environment via electronic mail.

You can also invoke any of the commands from this menu by typing keyboard commands into the listener itself. The precise commands you need to use depend on the Lisp image that you are running; see the appropriate manual for your image for more details.

16.9 Execute mode

The listener is actually a special type of editor window, which is run in a mode known as *execute mode*. This means that, as well as the normal keyboard commands available to the editor, a number of additional commands are available which are especially useful when working with an interactive prompt. Some of the commands available are as follows:

Press Alt+P or Ctrl+C Ctrl+P to display the previous event on the history list.

Press Alt+N or Ctrl+C Ctrl+N to display the next event on the history list (this is not available if you are at the end of the history list).

Press Alt+R or Ctrl+C Ctrl+R to perform a search of the history list.

These commands are very useful when you are repeating a series of similar commands several times. You can also use the **mathefactory** and **mathefactory** buttons on the LispWorks podium button bar to move up and down the listener history list.

For more details about other keyboard commands available in the editor, see Chapter 10, "The Editor", and the *Editor User Guide*.

16.10 Help with editing in the listener

Two help commands are available to provide you with more information about editor commands which can be used in the listener.

Choose **Help > Editing > Key to Command** and type a key sequence to display a description of the extended editor command it is bound to, if any.

Choose **Help > Editing > Command to Key** and supply an extended editor command to see the key sequence it is bound to, if any.

For more details about the keyboard commands and extended editor commands available, see Chapter 10, "The Editor". The Listener

17

The Output Browser

The output browser is a simple tool that displays the output generated by your programs, and by operations such as macroexpansion, compilation and tracing. You can create one by choosing **Tools > Output Browser**.

```
🛃 Output Browser 1
                                                                  ;;; Compiling file D:\src\lisp\common-lw\ib-example\ib-example.lisp .!
;;; Safety = 3, Speed = 1, Space = 1, Float = 1, Interruptible = 0
;;; Compilation speed = 1, Debug = 2, Fixnum safety = 3, GC safety = !
;;; Source level debugging is off
;;; Source file recording is on
;;; Cross referencing is off
; (TOP-LEVEL-FORM 1)
; (TOP-LEVEL-FORM 2)
; (SUBFUNCTION (DEFCLASS IB-EXAMPLE) (CAPI:DEFINE-INTERFACE IB-EXAMPL!
E))
; (CAPI:DEFINE-INTERFACE IB-EXAMPLE)
; (CAPI:DEFINE-INTERFACE IB-EXAMPLE)
; (SUBFUNCTION (METHOD CAPI::INITIALIZE-INTERFACE :AFTER (IB-EXAMPLE)!
) (CAPI:DEFINE-INTERFACE IB-EXAMPLE))
; (CAPI:DEFINE-INTERFACE IB-EXAMPLE)
; (TOP-LEVEL-FORM 3)
; Loading fasl file D:\src\lisp\common-lw\ib-example\ib-example.fsl
 --- Press space to continue ----
```

Figure 17.1 Output browser

The output browser has one main area that displays any textual output from the environment. Output mainly consists of compilation, and macroexpansion messages produced by the editor, but can also include output from other tools, such as the profiler. The main area is actually an editor window, so all the usual editor keyboard commands can be used in it. See Chapter 10, "The Editor" for more details about these operations.

The output browser is invaluable when you are developing code, because it can be used to collect any output generated by your code. An example of how to do this is given in Section 2.4 on page 11.

Many other tools in the environment contain an output view, which you can displayed by clicking the Output tab in the relevant tool. Such views are used to collect all the output which has been generated by that tool. For instance, the listener has an output view that displays all the output from forms evaluated in the listener, and the editor has an output view that collects any output generated by the editor, such as compilation messages or macroexpansions. Note that the output browser is the only tool which collects any output from your own code without any need for further action on your part. The Output Browser

The Process Browser

Each individual window in the Lisp environment runs as a lightweight process of the original UNIX process that started the image. The process browser gives you control over these lightweight processes. To create a process browser, choose **Tools > Process Browser** from the podium. Note that you cannot control the GC monitor from the process browser, since this runs as a separate UNIX process.

🛃 Process Browser	1			_ 🗆 🗵		
<u>F</u> ilter:			🖌 🗙 Mate	hes: 7		
Name	Priority	Status				
Inspector 1	600000	Waiting for	input input			
Editor 2 - foo.lisp	600000	Waiting for	events			
LispWorks 4.1.0	600000	Running				
Background execute 2 Background execute 1	500000	Waiting for Waiting for	job to execute iob to execute			
The idle process	-8388608	Running	1			
l						
Ready.						

Figure 18.1 Process browser

The process browser consists of a main area in which all the current processes in the environment are listed, and a Filter area which you can use to restrict the information displayed in the main area.

Like other filter areas in the environment, type a string in the Filter area and press Return to limit the display to only those items which contain the string. See Section 3.12 on page 38 for details on using the Filter area.

18.1 Displaying and sorting the output

The main area contains a list of all the current processes in the environment are listed. If you have many processes running, you can use the filter area to only list processes containing a given string. For example, if you enter "Running" in the filter area, and click on <u>,</u> only processes that have the word "Running" in their description will be shown.

The processes displayed in the main area can be sorted by clicking the relevant button above each column. For example, to sort all listed processes by name, click on the Name title button. The other two buttons are **Priority** and **Status**.

18.2 Process control

The **Works > Processes** menu contains commands that let you control the execution of processes in the environment. Process commands act on the process that has been selected in the main area. You can select a process by clicking on the line in the main area that contains the process name and status information.

Choose **Works > Processes > Break** to break the current process. This breaks Lisp and gives you the opportunity to follow any of the normal debugger restarts.

Choose Works > Processes > Kill to kill the selected process.

Choose **Works > Processes > Stop** to stop the selected process. The process can be started again by choosing **Works > Processes > Unstop**.

Choose Works > Processes > Unstop to restart a process which has been stopped using Works > Processes > Stop.

Choose **Works > Process > Inspect** to call up a LispWorks inspector to inspect the selected process. See Chapter 13, "The Inspector" for more information on inspecting objects and processes.

Finally, choosing **Works > Process > Remote Debug** breaks the selected process and puts it in the debugger. This this equivalent to selecting **Works > Process > Break** and then choosing **Debug** on the consequent notifier.

18.3 Other ways of breaking processes

If you do not wish to use the process browser to break a particular process, you can use the following keyboard command:

• Press Ctrl+Break to break the current process.

18.4 Updating the process browser

The process browser updates itself automatically when a new process is created and when a process terminates. Other changes such as processes sleeping and wake are not notices automatically so you may need to choose **Tools > Update** to view the latest status displayed for each process.

19

The Profiler

19.1 Introduction

The profiler provides a way of monitoring Lisp functions during the execution of source code. The data that you obtain can be used to improve the efficiency of your code by highlighting procedures which are commonly used or are particularly slow, and which would therefore benefit from extra optimization effort.

To create a profiler, choose **Tools > Profiler**.

The profiler has several areas. The Code to Profile panel lets you set up and profile any amount of Lisp source code.

- Use the large text box to paste or type in the Lisp source code that you wish to profile.
- The **Symbols** and **Packages** buttons let you choose which symbols and packages you want to profile.
- The **Profile** button starts code profiling.

The Results area is used to display the results of a profile.

• The main list displays the functions called during the last profile, together with statistical details such as the number of times each function was called and the number of times it was found on the stack.

- The Filter box lets you restrict the display of information in the Results area.
- The Profile Summary box provides a summary of the last profile performed.

🛃 Profile	r 1			_ 🗆 🗙
r Code To F	Profile ——			
[]capi:c (setq	ontain title ()	make-in:	stance 'capi:title-pane :text "Title")) <mark>)</mark>	▲ <u>S</u> ymbols <u>P</u> ackages ▼ Profile
⊢ Results —				
<u>F</u> ilter:				Matches: 100
0	6 (100%)) 0	LET	_
0 23 1084 9 2 476 15 114 Description Fu Lambd	3 (50%) 0 0 0 0 0 0 0 0 0 0 Name: nction: a List:	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	UNWIND-PROTECT OUTPUT-STREAM-P NUMBERP PACKAGE-TO-PACKAGE MAKE-LIST SPECIAL-FORM-P CHAR FOURTH FIND-CLASS	
Profile Sur profile Sur Drofile Sur	tation: Files: mmary: e-stacks stack n	called	6 times tored 100% of the time	▲ ▼

Figure 19.1 The profiler

You can add a description pane to the profile by clicking on **Description >>**. The Description area provides a description of any item selected in the Results

area, giving you the name, function, lambda list, documentation string and source files of the selected item.

19.2 A description of profiling

When code is being profiled, the Lisp process is interrupted regularly at a specified time interval. At each interruption, the profiler scans the execution stack and records the name of every function found. A special note is made of which function is at the top of the stack. When profiling stops (when the code being profiled has stopped execution) the profiler presents aggregated information about each function that includes the following information:

- The number of times each function being profiled was called.
- The number of times each function was found on the stack by the profiler, both in absolute terms and as a percentage of the total number of scans of the stack.
- The number of times each function was found on the top of the stack, both in absolute terms and as a percentage of the total number of scans of the stack.

The profiler gives you an easy way of choosing which functions you wish to profile, which code you want to run while profiling, and provides you with a straightforward display of the results of each profile.

19.3 Steps involved in profiling code

Each time you profile code, you first need to set up the profiler to ensure that you find out the sort of information you are interested in. This section gives you details about how to go about this.

The steps that you need to take when profiling code are as follows:

- 1. Choose which functions you want to profile.
- **2.** Specify the code that you want to run while profiling, and the package that you want to run the code in.
- **3.** Perform the profile.

Note: You do not have to adhere strictly to the sequence shown above, but this is the order that you should usually follow.

19.3.1 Choosing the functions to profile

It is possible to keep track of every function called when running code, but this involves significant effort in determining which functions are suitable for profiling and in keeping track of the results. To minimize this effort you should specify which functions you want to profile. The profiler checks that these functions have indeed got function definitions and are therefore suitable for profiling. For more information on the types of function that can be profiled, see Section 19.5.1.

There are two ways of specifying functions that you want to profile:

- Choose which individual functions you want to profile.
- Choose whole packages, all of whose functions are profiled.

19.3.1.1 Choosing individual functions

Click **Symbols** to specify a list of Lisp functions that you want to profile. The dialog shown in Figure 19.2 appears.

A Profiler 1	×
Select Symbols to Profile:	
New Symbol:	<u>0</u> K
Selected symbols:	All
EVAL	
MAKE-INSTANCE	<u>R</u> emove
UNWIND-PROTECT	
FUNCALL	Lancel

Figure 19.2 Select Symbols to Profile dialog

This dialog displays the list of functions to be profiled.

- To add a function to the list, type its name in the New Symbol text box and click
 I. To generate a completion list of symbols from which to choose, click
- To remove a function from the list, select it from the list and click **Remove**.
- To remove several functions, select them all before clicking **Remove**.

Click **OK** when you have finished choosing symbols.
19.3.1.2 Choosing packages

You may often want to profile every function in a package. Click **Packages** to specify a list of packages whose functions you want to profile. The dialog shown in Figure 19.3 appears.

🛃 Profiler 1		×
Select Packages to Profile:		
Package:	🖌 🏃	<u>0</u> K
Unselected Packages: CAPI CAPI-INTERNALS CAPI-LAYOUT CAPI-LIBRARY	Selected Packages: COMMON-LISP-USER COMMON-LISP LISPWORKS	<u>A</u> ll <u>N</u> one
CAPI-TOOLKIT CAPI-WIN32-LIB CHARACTER CLOS COLOR COLOR	>>>	
COMMONOTIONES COMPILER CONDITIONS CONSTANTS CORBA	<<<	
DBG DELIVERY DSPEC EDITOR ENVIRONMENT ENVIRONMENT-INTERN.		

Figure 19.3 Select Packages to Profile dialog

The main part of this dialog consists of two lists:

- The Unselected Packages list shows packages in the Lisp image whose functions are not to be profiled.
- The Selected Packages list shows packages in the Lisp image whose functions are to be profiled.

A global function definition is profiled if its symbol is visible in one of the Selected Packages.

In order to add a package to the Selected Packages list, you can do one of three things:

- For each package, type the package name in the Select Package box and press Return or click
- Select the package in the Unselected Packages list and click on >>>.
- Double-click on the package in the Unselected Packages list.

To remove a package from the Selected Packages list, do one of the following:

- Select the package in the Selected Packages list and click on <<<.
- Double-click on the package in the Selected Packages list.

In addition, there are four buttons.

Click **All** if you wish to profile all functions in all packages. Be aware that there are significant processing overheads involved if you wish to do this, and the results you get will probably include a lot of unwanted information.

Click **None** to clear the list of Selected Package. This is useful is you only want to profile a few functions, which you can specify easily using the **Symbols** button on the Profiler tool itself.

Click **OK** when you have finished selecting the packages whose functions you want to profile. The dialog disappears after clicking on this button.

Click Cancel to cancel the operation. This also dismisses the dialog.

19.3.2 Specifying the code to run while profiling

Code which is to be executed during profiling should be typed in the Code to Profile area. This is actually an editor window, and so you can use all the keyboard commands which can be used in the editor.

Code may be placed in this window in three ways:

• type it directly into the window

- paste it in from other editor windows in the environment
- paste it in from other applications

Specify the package in which you want to run the code to be profiled using the Package panel of the Profiler Preferences dialog. Choose **Tools > Preferences** to display this dialog. If you are unsure, full details on how to do this can be found in Section 3.7.1 on page 30. Like all other tools in the environment, the profiler can have a particular package associated with it; the default package is CL-USER.

19.3.3 Performing the profile

Once you have set up the profile as described above, perform the profile itself by clicking on the **Profile** button in the profiler.

19.4 Format of the results

After you have run the profile, a four column table is printed in the large list in the Results panel. These columns are laid out as follows:

Call#	The call count of each function, that is, the number of times it was called during execution of the code.
Stack#(%)	The number of times the function was found on the stack when the Lisp process was interrupted. The parenthesized figure shows the percentage of time the function was found on the stack.
Top#(%)	The number of times the function was found on the top of the stack when the Lisp process was interrupted. Again, the figure in brackets shows the percentage of time the function was found on top of the stack.
Name	The name of the function.

You can order the items in the list by clicking on the relevant heading button.

Selecting any item in the list displays a description of that function in the Description list. In addition, an item selected in the main list can be acted

upon by any relevant commands in the **Works > Function** menu. For instance, if you select a generic function in the main list and choose **Works > Function > Generic Function**, you can view the generic function in a generic function browser. This is consistent with many of the other tools in the environment.

Double-clicking on an item in the Description list invokes an inspector on the selected item. In addition, an item selected in this area may be acted on by any relevant commands in the **Works > Description** menu, as is the case with many other tools in the environment. For instance, choose **Works > Description > Copy** to copy the item selected in the Description list to the clipboard. See Section 3.8 on page 31 for details on the commands available.

19.5 Interpreting the results

The most important columns in the Results area are those showing call count (call#) and number of times on the top of the stack (stack#). Looking solely at the number of times a function is found on the stack (stack#) can be misleading, because functions which are on the stack are not necessarily using up much processing time. However, functions which are consistently found on the top of the stack are likely to have a significant execution time. Similarly the functions that are called most often are likely to have the most significant effect on the program as a whole.

Always remember that the numbers produced are from random samples, so you should be careful when interpreting their meaning. The rate of sampling is always coarse in comparison to the function call rate, so it is possible for strange effects to occur and significant events to be missed. For example, *resonance* may occur when an event always occurs between regular sampling times. In practice, however, this is not usually a problem.

19.5.1 Profiling Pitfalls

It is generally only worth profiling code which has been compiled. If you profile interpreted code, the interpreter itself is profiled, and this skews the results for the actual Lisp program.

Macros cannot be profiled because they are expanded during the compilation process. Similarly, the compiler may transform some functions such that they are present in the source code but not in the compiled code. For example, the compiler transforms this source expression:

(member 'x '(x y z) :test #'eq)

into this compiled expression:

(memq 'x '(x y z))

Therefore the function member cannot be profiled, since it is not called in compiled code.

Similarly, you cannot profile inlined functions.

Recursive functions need special attention. A recursive function may well be found on the stack in more than one place during one interrupt. The profiler counts each occurrence of the function, and so the total number of times a function is found on the stack may be greater than the number of times the stack is examined.

You must take care when profiling structure accessors. These compile down into a call to a closure, of which there is one for all structure setters and one for all structure getters. Therefore it is not possible to profile individual structure setters or getters by name.

Even if you configure the profiler to profile all the known functions of an application, it is possible that less than 100% of the time is spent monitoring the top function. This is because an internal system function could be on the top of the stack at the time of the interrupt.

19.6 Some examples

The examples below demonstrate different ways in which the profiler can be configured and code profiled so as to produce different sets of results. In each example, the following piece of code is profiled:

This is a simple piece of code which creates and displays a CAPI title pane; each time you run the code a small window is created on-screen, which you can ignore or remove as soon as it is created.

1. Create a profiler if you have not already done so.

- **2.** Type the code above into the box in the Code to Profile panel.
- 3. Choose Tools > Preferences to change the package of the profiler.

Notiler 1 Preferences	×
Sort ⓒ <u>Unsorted</u> ○ <u>By</u> Call Count ○ By <u>N</u> ame ○ By <u>O</u> ccurrences on Stack ○ By <u>O</u> ccurrences on <u>I</u> op of Stack ○ By <u>P</u> ackage ○ By <u>T</u> ype	Package COMMON·LISP-
0 <u>K</u>	Cancel Apply

Figure 19.4 Profiler Preferences dialog

- **4.** In the Profiler Preferences dialog, replace the default package in the Package text box with CAPI and click
- **5.** Click **OK** to dismiss the Preferences dialog and apply the change you have made.
- 6. Click on Profile.

This profiles the functions in the COMMON-LISP, CL-USER and LISPWORKS packages.

Next, add the CAPI package to the list of packages whose functions are profiled.

- 7. Click Packages.
- **8.** In the dialog, double-click on CAPI in the Unselected Packages list, and click on **OK**.
- **9.** Click on **Profile** to profile the code again.

Notice that this time there are many more functions which appear on the stack during profiling.

10. Select a few of the functions listed at the top of the Results area, and look at their function descriptions.

Notice that most of the functions appearing on the stack are in the CAPI package. It is worth profiling a few functions explicitly, and removing unwanted packages from the list of packages to profile.

11. Click **Symbols**, and add the following four functions to the list in the dialog:

funcall unwind-protect make-instance eval

Type the name of each function and press Return to add it to the list.

12. Click **OK** when you have finished adding to this list.

Now remove the unwanted packages from the list of packages to profile, as follows:

- 13. Click Packages.
- **14.** In the dialog, double-click on the following items in the selected Packages list:

COMMON-LISP COMMON-LISP-USER LISPWORKS

15. Click on OK, and profile the code again by clicking on Profile.

Notice that the four functions in the COMMON-LISP package are still being profiled, even though you are no longer profiling all functions from that package by default. The Profiler

The System Browser

20.1 Introduction

When an application becomes large, it is usually prudent to divide its source into separate files. This makes the individual parts of the program easier to find and speeds up editing and compiling. When you make a small change to one file, just recompiling that file may be all that is necessary to bring the whole program up to date.

The drawback of this approach is that it is difficult to keep track of many separate files of source code. If you want to load the whole program from scratch, you need to load several files, which is tedious to do manually, as well as prone to error. Similarly, if you wish to recompile the whole program, you must check every file in the program to see if the source file is out of date with respect to the object file, and if so re-compile it.

To make matters more complicated, files often have interdependencies; files containing macros must be loaded before files that use them are compiled. Similarly, compilation of one file may necessitate the compilation of another file even if its object file is not out of date. Furthermore, one application may consist of files of more than one source code language, for example Lisp files and C files. This means that different compilation and loading mechanisms are required.

The Common LispWorks system tools, and the system browser in particular, are designed to take care of these problems, allowing consistent development and maintenance of large programs spread over many files. A system is basically a collection of files that together constitute a program (or a part of a program), plus rules expressing any interdependencies which exist between these files.

You can define a system in your source code using the defsystem macro. See the User Guide for more on the use of defsystem. Once defined, operations such as loading, compiling and printing can be performed on the system as a whole. The system tool ensures that these operations are carried out completely and consistently, without doing unnecessary work, by providing you with a GUI front end for defsystem.

A system may itself have other systems as members, allowing a program to consist of a hierarchy of systems. Each system can have compilation and load interdependencies with other systems, and can be used to collect related pieces of code within the overall program. Operations on higher-level systems are invoked recursively on member systems.

20.2 Defining a system

A system is defined with a defsystem form in an ordinary Lisp source file. This form must be evaluated in the Lisp image in order to define the system in the environment. Once loaded, operations can be carried out on the system by invoking Lisp functions, or, more conveniently, by using the system browser.

For example, the expression:

CL-USER 5 > (compile-system 'debug-app :force t)

would compile every file in a system called debug-app.

Note: When defining a hierarchy of systems, the leaf systems must be defined first—that is, a system must be defined before any systems that include it.

By convention, system definitions are placed in a file called defsys.lisp which usually resides in the same directory as the members of the system.

The full syntax of defsystem is given in the *LispWorks Reference Manual*, and further examples and a summary are given in the *LispWorks User Guide*.

20.2.1 Examples

Consider an example system, demo, defined as follows:

```
(defsystem demo (:package "USER")
  :members ("macros"
        "demo-utils"
        "demo-functions")
  :rules ((:in-order-to :compile ("child1" "child2")
        (:caused-by (:compile "parent"))
        (:requires (:load "parent")))))
```

This system compiles and loads members into the USER package if the members themselves do not specify packages. The system contains three members—macros, demo-utils, and demo-functions—which may themselves be either files or other systems. There is only one explicit rule in the example. If macros needs to be compiled (for instance, if it has been changed), then this causes demo-utils and demo-functions to be compiled as well, irrespective of whether they have themselves changed. In order for them to be compiled, parent must first be loaded.

Implicitly, it is always the case that if any member changes, it needs to be compiled when you compile the system. The explicit rule above means that if the changed member happens to be macros, then *every* member gets compiled. If the changed member is not macros, then macros must at least be loaded before compiling takes place.

The next example shows a system consisting of three files:

```
(defsystem my-system
 (:default-pathname "~/junk/")
 :members ("a" "b" "c")
 :rules ((:in-order-to :compile ("c")
                     (:requires (:load "a"))
                     (:caused-by (:compile "b")))))
```

What plan is produced when all three files have already been compiled, but the file **b.lig** has since been changed?

First, file a.lisp is considered. This file has already been compiled, so no instructions are added to the plan.

Second, file b.lisp is considered. Since this file has changed, the instruction *compile b* is added to the plan.

Finally file c.lisp is considered. Although this has already been compiled, the clause

(:caused-by (:compile "b"))

causes the instruction *compile c* to be added to the plan. The compilation of c.lisp also requires that a.lisp is loaded, so the instruction *load a* is added to the plan first. This gives us the following plan:

- 1. Compile b.lisp.
- 2. Load a.lisp.
- 3. Compile c.lisp.

20.3 A description of the system browser

The system browser provides an intuitive way to examine systems which have been loaded into the environment, together with their members. You can use it to perform system-wide actions, thereby creating plans which you can review before executing. The system browser has a total of four views available:

- The graph view is used to display a graph of all the requested systems defined in the image, together with their members.
- The text view lists the systems defined in the image together with the members of the current system.
- The preview view provides a powerful way of generating and executing systems plans.
- The output view is used to display any output messages which have been created by the system browser as a result of executing plans.

To create a system browser, choose Tools > System Browser from the podium. Alternatively, choose File > Browse Parent System from any appropriate tool in the environment or execute Alt+x Browse System in an editor, to display the parent system for the selected or current file in the system browser. See Section 3.6 on page 28 for details.

20.4 Examining the system graph

When you first invoke the system browser, the graph view is the default view. You can also switch to it from another view by choosing the relevant tab above the main view. The graph view is shown in Figure 20.1 below.

♪ System Browser 1	_ 🗆 ×
System: ROOT-SYSTEM	🗸 🗙 🐉
Text Graph Preview Output	
ROOT-SYSTEM	
Description: Module: ROOT-SYSTEM Pathname: Z:\lw-4-1\pcl\ Flags:	
Ready.	

Figure 20.1 Displaying loaded systems using the graph view

The system browser window has four areas, described below.

20.4.1 System area

The *System area* is used to type in the name of the system you want to examine.

In order to browse a given system:

1. Load the Lisp source code defining the system into the environment.

For instance, select the file in the file browser or display it in the editor and choose File > Load. Alternatively, choose File > Load from the system browser and choose a file to load in the dialog that appears.

2. Type the name of the system into the System area.

You can press the Tab key at any point in your typing to complete the name, or display a dialog listing the possible completions, as appropriate.

The members of the system are displayed in the graph area.

If you wish to browse the parent system of the current system, choose **Works > Systems > Parent**. The parent of all systems defined in the image at any time is called the ROOT-SYSTEM.

20.4.2 Graph area

The *Graph area* produces a graph of the current system, together with all its members. The generic facilities available to all graph views throughout the environment are available here; see Chapter 5, "Manipulating Graphs" for details.

- Double-click on a file to display it in the editor.
- Double-click on a system to display its members in the system browser.
- Select either a system or a file to display details in the Description area.

Items selected in this area can be operated on by commands in the **Works > Systems** menu. If no items are selected, the commands apply to the current system, whose name is printed in the System area.

20.4.3 Description area

The *Description area* shows details about any system member selected in the Graph area. The following items of information are shown:

Module	The name of the selected member. This is either the file- name (if the member is a file of source code) or the sys- tem name (if the member is a subsystem).
Pathname	The directory pathname of the selected member. This is the full pathname of the file, if the selected member is a file of source code, or the default directory of the sys- tem, if the selected member is a subsystem.
Flags	This lists any keyword flags which have been set for the selected member in the system definition, such as the <code>:source-only</code> flag.

To operate on any of the items displayed in this area, select them and choose a command from the **Works > Description** menu, which contains the standard actions described in Section 3.8 on page 31. By making multiple selections, you can operate on as many of the items as you like.

20.4.4 Performing operations on system members

A variety of operations can be performed on any number of nodes selected in the Graph area. If no nodes are selected, or if you are in another view, the commands are performed on the current system, whose name is printed in the System area.

The **Works > Systems** menu gives you access to the standard actions described in Section 3.8 on page 31.

Choose **Works > Systems > Parent** to browse the parent system. This takes you up one level in the hierarchy.

Choose Works > Systems > Compile and Load, Works > Systems > Compile, or Works > Systems > Load to compile or load the selected members and any subsystems selected in the Graph area. Choose **Works > Systems > Concatenate** to produce one fasl file from several fasl files.

Choose **Works > Systems > Search Files** to search any of the selected members (and any subsystems) for a given string. The matches are displayed in the editor.

Choose **Works > Systems > Replace** to search all the files in the selected members (and any subsystems) for a given string and replace it with another string. You are prompted for a search string a string to replace it with in the echo podium.

20.5 Examining systems in the text view

The text view allows you to list the parent system, subsystems and files in the current system in one view, and gives you an easy way of changing the current system. Choose the Text tab to display this view. The system browser appears as shown in Figure 20.2 below.

🛃 System Browser 1	
System: DEMO	🖌 🗙 🐉
Text Graph Preview Output	
Parent System:	Subsystems:
ROOT-SYSTEM	
Files:	
example.lisp hello.lisp	
Description:	
Module: DEMO Pathname: z:\clc\not-in-hope\demo\ Flags:	
Ready.	

Figure 20.2 Displaying loaded systems using the text view

The system browser contains the areas described below when in the text view.

20.5.1 System area

As with the graph view, the current system is shown here. See Section 20.4.1 on page 254 for details about this area.

20.5.2 Parent system area

This area lists any parent systems of the current system. Note that every system apart from the ROOT-SYSTEM must have at least one parent.

Double-click on any item in this list to make it the current system. Its name is printed in the System area.

20.5.3 Subsystems area

This area lists any systems which are subsystems of the current system.

Double-click on any item in this list to make it the current system. Its name is shown in the System area.

20.5.4 Files area

This area lists any files which are members of the current system. Source files containing either Lisp or non-Lisp code (such as C code which is loaded in via the foreign language interface) are listed in this area.

- Select a file to display its description in the Description area.
- Double-click on a file to display it in the editor.

20.5.5 File description area

The File Description area displays information about any system member selected in the Files area. If no such member is selected, information about the current system (the one named in the System area) is shown instead. The same pieces of information are shown as in the graph view. See Section 20.4.3 on page 255 for details. As with other views, items selected in this area can be operated on using commands in the **Works > Description** menu.

20.6 Generating and executing plans in the preview view

The preview view allows you to generate different system plans automatically based on three things:

- the current compilation and load status of each member of a system
- the rules specified in the system definition
- the specific actions that you wish to perform

You can use this view to browse the plan and to execute all or any part of it, as well as generate it.

Click on the Preview tab to switch to the preview view in the system browser. The system browser appears as in Figure 20.3 below.

🛃 System Browser 1					_ 🗆 ×
<u>S</u> ystem: DEMO				•	< 🗙 🔅
Text Graph Preview	Output				
Filter:			 	🗙 Match	ies: 5
Compile system DEMO inf Compile hello.lisp Load system DEMO into Load example.fs Load hello.fsl	© COMMON-LISP-USE	R R			
Recompute Events	Execute Events	Actions: 🔽	Compile	🔽 Load	Force
Ready.					

Figure 20.3 Previewing system plans using the preview view

The system browser has the areas described below.

20.6.1 System area

As with the graph view, the current system is shown here. See Section 20.4.1 on page 254 for details about this area.

20.6.2 Actions area

The *Actions area* contains a number of options allowing you to choose which actions you want to perform, thereby allowing you to create system plans.

The **Compile**, **Load** and **Force** buttons can be switched on and off as desired. Note that at least one of **Compile** and **Load** must always be selected.

- Select **Compile** to create a plan for system compilation. The plan displays what actions need to be performed in order to create an up-to-date compilation of the entire system.
- Select Load to create a plan for loading the system. The plan displays a list of the actions required to load an up-to-date version of the system into the environment.
- Select **Force** if you want to force compilation or loading of all system members, whether it is necessary or not.

Click **Recompute Events** to create a new plan for the specified options. You should click this button whenever you change the **Compile**, **Load**, or **Force** options, or whenever you change any of the files in the system or any of its subsystems.

Click **Execute Events** to execute the events currently selected in the main area. See Section 20.6.6 below for details.

20.6.3 Filter area

As with other tools, you can use the Filter area to restrict the output in the plan area to just those actions you are interested in. This may be useful, for instance, if you want to see only compile actions, or only load actions, or if you are only interested in the actions that need to be performed for a particular file.

20.6.4 Plan area

The Plan area lists the actions in the current plan. Items are indented to indicate groups of related actions. Thus, if a subsystem needs to be loaded, the individual files or subsystems that comprise it are listed underneath, and are indented with respect to it.

20.6.5 File description area

The File Description area displays information about any system member selected in the Plan area. If no such member is selected, information about the current system (the one named in the System area) is shown instead. The same pieces of information are shown as in the graph view. See Section 20.4.3 on page 255 for details. As with other views, items selected in this area can be operated on using commands in the **Works > Description** menu.

20.6.6 Executing plans in the preview view

Once you have created a plan in the preview view, there are a number of ways that you can execute either the whole plan, or individual actions within that plan.

As already mentioned, to execute individual actions in the plan, select them in the main area and then click **Execute Events**.

To execute the whole plan, just choose the relevant menu command:

- Choose Works > Systems > Load if you want to execute a plan for loading the system.
- Choose Works > Systems > Compile if you want to execute a plan for compiling the system.
- Choose Works > Systems > Compile and Load if you want to execute a plan for both compiling and loading the system.

Note that you can also execute the whole plan by choosing Edit > Select All and then clicking Execute Events.

20.7 Examining output in the output view

The output view can be used to view any messages that have been generated as a result of actions performed in the system browser. This largely consists of compilation and load messages that are generated when system plans or individual actions in a plan are executed.

Click on the Output tab to switch to the output view. The system browser appears as in Figure 20.4.



Figure 20.4 Viewing output in the system browser

The output view has the areas described below.

20.7.1 System area

As with the graph view, the current system is shown here. See Section 20.4.1 on page 254 for details about this area.

20.7.2 Output area

The largest area in this view is used to display all the output messages which have been generated by the system browser. This area has the same properties as the output browser described in Chapter 17, "The Output Browser".

20.7.3 File description area

The File Description area displays information about the last system member selected in another view. If no such member has been selected, information about the current system (the one named in the System area) is shown instead. The same pieces of information are shown as in the graph view. See Section 20.4.3 on page 255 for details. As with other views, items selected in this area can be operated on using commands in the **Works > Description** menu.

20.8 Configuring the display

The system browser allows you to configure the display so that it best suits your needs. The commands available for this are described below.

20.8.1 Sorting entries

Entries in the system browser can be sorted in a number of ways. Choose **Tools > Preferences** to display the System Browser Preferences dialog. Click on the General tab to view the sorting options.

By Name	Sorts entries in the main area of the current view (the graph in the graph view and the Files area in the text view) according to the symbol name.
By Paclage	Sorts entries in the main area according to their pack- age.
Unsorted	Leave entries in the main area unsorted. This is the default setting.

20.8.2 Displaying package information

As with other tools, you can configure the way package names are displayed in the system browser, using the Package panel. See Section 3.7 on page 29 for full details.

20.9 Setting options in the system browser

The **Works > Systems** menu allows you to set options which apply whenever you compile or load system members. Each of the commands described below toggles the respective option.

Choose **Works > System> Compilation Options > Force** to force the compile or load operation to be performed. If you are operating on a whole system (as opposed to system members which are files) this means that actions for all the members are added to the plan.

Choose **Works > System> Compilation Options > Source** to force the use of Lisp source rather than fasls in operations on the system.

Choose **Works > System> Compilation Options > Preview** to automatically preview the plan prior to execution of a compile or load instruction chosen from the **Works > Systems** menu. This switches the system browser to the preview view and allows you to see what operations are going to be performed, and to change them if you want. See Section 20.6 on page 258 for full details about previewing plans.

Choose **Works > System > Concatenate** to concatenate the selected system into a single fasl after compiling it.

The Window Browser

21.1 Introduction

The window browser lets you examine any windows that have been created in the environment. You can examine not only the environment windows themselves, but also more discrete components of those windows, such menus and menu commands. To create a window browser, choose **Tools > Window Browser** from the LispWorks podium.

The window browser only has one view, shown in Figure 21.1.

🛃 Window Bro	owser 1	_ 🗆 ×
Graph: # <lispw< td=""><td>ORKS-TOOLS::APPLICATION "LispWorks" 20617AFC></td><td></td></lispw<>	ORKS-TOOLS::APPLICATION "LispWorks" 20617AFC>	
"LispWorks"•—		×
•		
Description:		
Window: Class: Name: Representation: Interface: Screen:	# <lispworks-tools::application "lispworks"="" 20617afc=""> LISPWORKS-TOOLS::APPLICATION</lispworks-tools::application>	
Ready.		

Figure 21.1 The window browser

The window browser has three sections.

21.1.1 Graph box

This text box shows the window object that is being examined; that is, the name of the window at the root of the graph.

21.1.2 Window graph

The window graph displays the current window and all its subwindows. The generic facilities available to all graphs throughout Common LispWorks are available here; see Chapter 5, "Manipulating Graphs" for details.

When you first create a window browser, it automatically browses the parent window of the whole environment. A graph of the parent window together with its children—each individual window that has been created—is drawn in the main area.

Select any item in the graph to display its description in the Description area.

To examine any child in the graph, double-click on it. The child is expanded in the current graph. Choose **Works > Windows > Browse - Window** to make the selected child the root of the graph.

Any items selected in the graph can be operated on using commands in the **Works > Windows** menu. If no items are selected, the commands in this menu apply to the root window of the graph. See Section 21.3 on page 270 for details.

21.1.3 Description list

The Description list gives a description of the item selected in the Graph area. If nothing is selected, a description of the window at the root of the graph is shown. The following information is listed:

Window	The object which represents the selected window
Class	The class of the window object.
Name	The name of the selected window.
Representation	The CAPI representation of the selected window.
Interface	The underlying native window system object which represents the selected window.
Screen	The name of the screen on which the selected window is displayed.

Any item selected in the Description list can be operated on by using commands under the **Works > Description** menu. This menu gives you access to the standard actions commands described in Section 3.8 on page 31.

21.2 Configuring the window browser

Using the Window Browser Preferences dialog, shown in Figure 21.2, you can configure the window browser so that it best suits your needs. Choose **Tools > Preferences** to display this dialog.

🛃 Window Browser 1	Preferences X
General Graph Layout Display Component Layouts	Components
Pinboard Objects Menu Items	Menus
 Short Names 	O Long Names
	K <u>C</u> ancel Apply

Figure 21.2 Window Browser Preferences dialog

The Window Browser Preferences dialog has three tabs:

- The General tab contains options for configuring general properties of the window browser.
- The Graph Layout tab contains options for configuring options specific to the graph. See Section 5.6 on page 50 for a description of these options.
- The Components tab contains options for configuring properties unique to the window browser.

21.2.1 Sorting entries

Entries in the window browser can be sorted using the Sort panel in the General tab of the Preferences dialog. Choose the sort option you require from the list available.

By Name	Sorts items alphabetically by name.
By Package	Sorts items alphabetically by package name.
Unsorted	Displays items in the order they are defined in. This is the default setting.

21.2.2 Displaying package information

As with other tools, you can configure the way package names are displayed in the window browser using options available in the General tab.

Check or un-check **Show Package Names** to turn the display of package names in the window browser on and off.

Specify the process package of the window browser in the appropriate text box.

21.2.3 Displaying different types of window

There are several types of window object which can be displayed in the window browser, and you can configure which types are displayed using the Display Component panel of the Components tab in the Preferences dialog. Five options are available; select whichever ones you want to display.

Layouts Displays the major layouts available to the window being graphed. For the parent window of the environment, this means all the windows that have been created. For an individual window, this means the configuration of the different panes in that window.

Subwindows	Displays any subwindows for the current window. This option only takes effect if the current window is the parent of a group of windows, rather than a real win- dow itself.
Pinboard Objec	tsDisplays any pinboard objects in the current window. See the <i>CAPI User Guide</i> for a full description of pin- board objects.
Menus	Displays any menus available to the current window in the graph.
Menu Items	Displays any menu items available to the current win- dow in the graph. This option only takes effect if Menus is selected as well.

By default, all five options are already selected in the window browser.

21.2.4 Displaying short or long names

By default, the window browser gives each item in the graph a short name. You can also display the complete symbol name for each item if you wish, as displayed in the Window line of the Description list. You can configure this option from the Components tab of Preferences dialog.

Click **Long Names** to display the complete symbol name of each item in the graph.

Click **Short Names** to display the short name for each item in the graph. This is the default setting.

Bear in mind that graphs are a lot larger when you display them using long names, and can therefore be more difficult to examine.

21.3 Performing operations on windows

You can perform a number of operations on any windows selected in the Graph area using the commands in the **Works > Windows** menu. If no items are selected in the Graph area, the commands in this menu apply to the root window of the graph.

The **Works > Windows** menu gives you access to the standard actions commands described in Section 3.8 on page 31.

21.3.1 Navigating the window hierarchy

Choose **Works > Windows > Browse Parent** to display the parent of the current window. This takes you back up one level in the window hierarchy.

Choose **Works > Windows > Browse Representation** to examine the representation of the current window. This is discussed in Section 3.13 on page 39.

Choose **Works > Windows > Browse Screens** to examine the parent window of the environment once again—this takes you back up to the root of the window hierarchy.

21.3.2 Window control

There are several commands which give you control over the current window.

Choose **Works > Windows > Lower** to push the current window to the bottom of the pile of windows on-screen.

Choose **Works > Windows > Raise** to bring the current window to the front of your screen.

Choose Works > Windows > Quit to quit the any windows selected in the graph.

Choose **Works > Windows > Destroy** to destroy any windows which are selected in the graph. You are prompted before the windows are destroyed. The Window Browser

Index

Symbols

\$ variable 168
* variable 12, 17, 166, 220
** variable 12
*** variable 12
.lispworks file 24

A

aborting commands in the editor 122 action callbacks 184 Actions menu **31–34** Documentation 102 Find Definition 102 Alt key use of 111 Arguments command 133, 222 Attributes command 210, 213

B

Backtrace command 224 backtraces 100 binding \$ to the current inspector object 168 binding frames 103 Bindings button 103 **Bindings command 224** Break command 233 Break on Access command 163 Break on Read command 163 Break on Write command 163 breaking a process 233 Browse command 32, 39, 267 variations in name 33 Browse Metaclass command 70, 74, 76, 80, 83,85

Browse Parent command 271 Browse Parent System command 28, 252 Browse Representation command 271 Browse Screens command 271 browsing Common Lisp classes 59–85 compilation conditions 87-92 errors 87 function calls 135-144 generic functions 145–154 native window system 271 online manuals 42 output 11, 227-229 selected object, class of the 33, 222 systems 28, 252-264 window definitions 265-271 buffers closing 120 swapping between 120 Buffers menu 114, 130 Compile 131 Evaluate 131 Trace. See Trace menu Undefine 133 bugs, reporting 97, 224 By Name option 36 By Package option 36

С

call frames 99, 103 callbacks action 184 extend 184 retract 184 selection 184

specifying 183-186 catch frames 103 Catchers button 103 :center keyword 179 Change Hidden Packages command 104 check components 203 choosing menu commands xi class browser 59-85 Class area 69 current class, operations specific to the 70, 74, 76, 80, 83, 85 description 3 Description area 73, 76, 82 examining a class 65 Filter area 69 filtering information 63 Function description area 79 functions list 79 functions view 78-80 generic functions, operating on 80 Graph area 76 graph view 74-77 hierarchy view 65 Include Inherited Methods/Functions button 79 inherited slots 62 Initargs area 82 initargs view 81 invoking on the current expression 222 invoking on the selected object 33 menu commands, see menu or command name overview of the 59 Precedence area 85 precedence view 83 Slot description area 70 slot information 62-64 Slots area 69 slots view 62-64 sorting information 66 tracing classes from the 80 undefining functions and methods 80 See also classes Class command 15, 33, 60, 63, 69, 222 classbrowser methods list 79 classes 59-85 changing slot values in the inspector 163-166 column-layout 198 displaying graphs of 74–77

examining 65 examining functions and methods defined on 78-80 inherited slots in 62 initargs 81 inspecting local slots 158 interface 193 list-panel 63 operations specific to the current class 70, 74, 76, 80, 83, 85 precedence list 83 push-button-panel 65 row-layout 198 tracing 80 See also class browser Classes menu Browse Metaclass 70, 74, 76, 80, 83, 85 objects operated on by the 70, 74, 76, 80, 83.85 clipboard general use 25-26 usage in editor 122 See also clipboard, kill ring, UNIX clipboard Clone command 22, 23, 37, 121 Close command 121 interface builder 207 closing editor buffers 120 Collapse Nodes command 48 collapsing graphs 47 column-layout class 198 Command to Key command 134, 225 commands completion of 110 repeating 26 common features in the environment 19-See also under graphs Common Lisp classes. See classes debugging 93-105 displaying documentation for expressions 222 displaying documentation for selected object 33, 102 evaluating forms 219-220 file extension 117 indentation of forms in source code 133 prompt 218 systems. See system compilation conditions browser 87-92

pathnames 91 preference dialog 90 types to display 91 **Compilation Options menu** Force 264 Preview 264 Source 264 Compile and Load command 28, 99, 132, 255, 261 Compile command 28, 131, 132, 215, 255, 261 Compile Region command 223 in editor 131 compiler output 131 compile-system function 250 compiling code editor 131-132 interface builder 215 compiling files in the listener 28 completion in class browser 65 Concatenate command 256. 264 Condition menu Actions. See Actions menu Report Bug 101 confirmer description 214 Confirmer command 214 consistency in the environment. See common features in the environment contain function 13.63.160.166 Contents radio button 43 Control key, use of xii, 111 conventions used in the manual ix-xii Copy command 25-26, 33, 123, 124, 165 in Actions menu 244 interface builder 209 standard action command 33 copying windows 22 creating new files 28, 117 current object. See selected object package of any tool 30 prompt 219 value, operating on 223 current class, operations specific to the 70, 74, 76, 80, 83, 85 current expression displaying lambda list for 133, 222 displaying value 133, 222 tracing 222 current form

macroexpanding 223 walking 223 Customize menu Reuse 22 Reuse all tools **21** Cut command **25–26**, 123, 124 interface builder 176, **209**

D

Debug menu Debugger 9, 94, 94, 97, 100, 101 Listener 9 Restarts 224 debugger 93-105 abort restart 98 binding frames 103 call frames 99, 103 catch frames 103 continue restart 98 controlling from the listener 223-224 debugger tool 224 description 97 displaying documentation for object in current frame 102 example session 99-101 finding source code for object in current frame 102 handler frames 103 invisible frames 104 invoking 95 invoking from the tracer 37 menu commands in the listener 223 restart frames 104 restart options 97–98 stack 98-99 stack frames 98 See also debugger tool Debugger command 9, 94, 94, 97, 100, 101, 224 debugger tool 93-105 Backtrace area 96 buttons 96 Condition area 96 invoking 94 invoking from notifier 96 package information 105 Restarts menu 97 types of frame, displaying 103 Variables area 96 See also debugger Definitions menu 115, 130 Compile 131

Evaluate 131 Generic Function 133 Trace. See Trace menu Undefine 133 defsystem macro 252 examples of use 251 deleting text in the editor 120 See also kill ring description of compilation conditions 89 Description menu 89, 139, 148 Deselect All command 26, 122 Destroy command 271 Dialog command 214 **Display Borders command 214** display function 189 displaying package information 29-31 windows 20 display-message function 187 Documentation command 33, 102, 222 documentation, online. See online help \$ variable 168

Е

Each tool has its own menu bar 22 Edit menu 24 Copy 25, 33, 123, 124, 209 Cut 25, 123, 124, 176, 209 Deselect All 26, 122 Find **26**, 127, 129 Find Next 26. 127 Find Next, for graph view 47 Find, for graph view 47 interface builder 208 Link. See Link Menu Paste 25, 69, 123, 147, 176, 209 Replace 26, 128, 129 Select All 26, 122, 261 Undo 25, 122 Editing menu Command to Key 134, 225 Key to Command 134, 225 editing the history list 27 editor aborting commands 122 buffers view 113, 120 buffers. See buffers changed definitions list 115 closing buffers 120 compiling source code 131–132 creating a new window 121

creating files 28, 117 current expression, displaying value 133 current package and displayed package 117 definitions list 114 definitions, operating on 133–134 deleting text 120 EMACS, comparison with 107 evaluating source code 130–131 expressions, operating on 133-134 History menu 120, 121 indenting forms 133 inserting files into the current buffer 120 inserting text 120 invoking 108 keyboard commands, use of 111 kill ring. See kill ring Lisp-specific commands 129–134 macroexpanding forms in the 133 menu and keyboard commands, distinctions 117 menu commands. See menu or command name moving around in the buffer 118-119 new files 28. 117 online help 134, 225 opening files 28, 108, 117 output view 115 overview 2 package information 116 package usage 132 repeating commands 122 replacing text 128–129 reverting to last saved version 118 saving files 117, 121 saving text regions 118 scrolling text 119 searching 127-128 sorting entries 116 swapping between buffers 120 tracing 132 undefining symbols 133 undoing commands 122 using the clipboard 122 viewing two sections of the same file 121 views available 108 walking forms 134 Editor command 21, 108 editor commands finding keyboard command for 225 Visit Tags File 111 EMACS
comparison with built-in editor 107 Enter Search String dialog 42 environment common features 19-40 quitting 24 error conditions 89 Escape key, use of **xii**, 111 **Evaluate command 215** in editor 131 Evaluate Region command in editor 131 in listener 222 evaluating code in the editor 130–131 forms 219-220 event next 224 previous 224 repeating 27 examining objects 157 execute mode 224 Exit command 6 Expand Nodes command 48 expanding graphs 47 Expression menu 130 Arguments 133, 222 Class 222 Compile Region 131, 223 **Documentation 222** Evaluate Region 131, 222 Find Source 222 Generic Function 223 Macroexpand 133 Macroexpand Form 223 Trace. See Trace menu Value 133, 222 Walk 134 Walk Form 223 expressions browsing the class of 222 displaying documentation 222 finding source code 222 extend callbacks 184 extended editor commands, finding keyboard command for 225 :extended-selection keyword 179

F

File menu 6, 58 Browse Parent System 28, 252 Close 121, 207 Compile 28, 132, 215

Compile and Load 28, 99, 132 description 28 Exit 24 Insert 120 Load 28, 131, 254 New 28, 117, 175, 189, 193 Open 28, 108, 117, 194 Print 28, 118 Revert 118, 207 Save 117, 186, 189, 207 Save All 121, 207 Save As 118, 207 Save Region As 118 filenames completion of 110 extensions for CL files 117 files compiling in listener 28 creating new 28, 117 inserting one into another 120 loading 28 navigating in the editor 118-119 opening 28, 108, 117 printing 28 reverting to last saved version 118 saving 117 saving all 121 filtering information 38-39, 63, 260 in inspector 160 Find command 26 in editor 127, 129 in graph view 47 Find Definition command 102 Find Next command 26, 127 in graph view 47 Find Source command 33, 222 in Debug menu 224 shortcut in debugger tool 100 Force command 264 forms compiling in editor 131–132 evaluating 219-220 evaluating in editor 130-131 indentation of 133 re-evaluating 220-221 Frame menu Find Source 100 Restart Frame 102, 104 Return From Frame 102 Trace. See Trace menu function call browser 135-144 By Name command 142

By Package command 142 Callees area 140 Callers area 141 description 135 Documentation area 139 Function area 136 Function description area 138 Function menu 144 Trace submenu 144 Graph area 137 graphing callers and callees 137 invoking on selected object 34 menu commands, see menu or command name operating on functions 144 package information 142 Show Package Names command 142 sorting entries 142 text view 139 tracing from 144 Unsorted command 142 views available 135 Function Calls command 34 Function menu 147 in the profiler 244 Trace. See Trace menu functions compile-system 250 contain 13, 63, 160, 166 display 189 display-message 187 undefining 80 Functions menu in the class browser 80 in the function call browser 144

G

generic function browser 145–154 Arguments types area 152 description 145 Description area 148 displaying signatures 153 Filter area 147 Function area 146 invoking on selected object 34, 133, 223 menu commands. *See* menu or command name Method combination list 152 method combinations, viewing 149 methods list 147 operating on signatures 153

Signatures area 151 Generic Function command 34, 133, 223, 244generic functions browsing from listener 223 in class browser 80 defined on selected object 34, 133 get-inspector-values 169 global preferences setting 23 graph layout menu 45 Collapse Nodes 48 Expand Nodes 48 Force Re-layout of Graph 55 Preferences 50 graph view system browser 253-255 graphical user interface. See interfaces graphs 45-55 altering breadth 51 altering depth 51 children function 185, 188 different layouts 53-55 expanding and collapsing nodes 47, 185 menu commands. See menu or command name searching 47 sorting items 35 GUI. See interfaces

Η

handler frames 103 Handlers button 103 Help menu 41, 58 Editing. See Editing menu Manuals 42 On Symbol 43 On Tool 43 Search 42 help. *See* online help Hidden Symbols button 103 hierarchy view in class browser 65 history list 26 editing the 27 in the listener 221 repeating next event 27, 224 repeating previous event 27, 224 searching the 224 History menu 27 in editor 120, 121 in the listener 221

interface builder 194 in listener 221–222 Modify 27 Next 27 Previous 27 HTML browsers use for online help 41

I

Include Inherited Slots button 62, 69 Include Inherited Slots checkbox 16 incremental search 127 Index radio button 43 initargs of slot, displaying 70 initform of slot, displaying 70 initialization file 24 Insert command 120 inserting files in editor 120 inserting text in editor 120 Inspect command 13, 23, 33, 157, 160, 162, 164, 233 inspecting listener values automatically 168 inspector 155–172 changing values 163-166 description 155 display options 166-168 filtering display 160 inspecting selected object 33 menu commands. See menu or command name overview 3 simple use 157 sorting entries 167 tracing 163 tracing in the 163 viewing local class slots 158 **Inspector command** 155 interface builder 191-215 adding your own code 204 attribute categories 212-213 Attributes dialog box 178, 181, 210 Attributes to View dialog box 184 button panels 197 Check Component button 204 code area 206 code view 183, 205 Component button 202 current interface 197 default names of elements 177, 199 default names of menus 201 Edit menu 208

editing code 205 example of use 173-190 interface area 197 Interface menu 213-215 interfaces, creating 175, 193–194 interfaces, loading 194-195, 208 introduction 191 invoking 192 Item button 181, 201 layout hierarchy area 175, 197 layouts view 175-177, 196-199 Menu Bar button 201 Menu button 181. 201 menu hierarchy area 175, 201 menus view 180-183, 199-204 methods of use 204 **Options button** 184 Radio Component button 203 rearranging components 176-177, 209-210 saving code 186, 189, 207-208 setting attributes 177-180, 210-213 switching between interfaces 194 Update button 207 views, description 191 See also interfaces interface class 193 Interface menu 39 Attributes 213 Confirmer 214 Dialog 214 **Display Borders 214** Evaluate 215 interface builder 213-215 Raise 213 Regenerate 214 interface skeleton default menus in 193 description 193 interfaces callbacks 183-186, 212 compiling 215 confirmers 214 constructing 175-177 creating menus for 199-204 creating new 175, 193-194 definition 193 development strategy 204 dialog boxes 214 evaluating 215 geometry of elements 213 graph area 197

layout elements, adding 198 layout hierarchy 197 loading **194–195**, 208 menu hierarchy 201 operating on the current 213–215 rearranging components 176–177, **209–210** regenerating 214 setting attributes 177–180, 210–213 titles 177–179, 212 types of attribute 212–213 invisible frames 104 Invisible Functions button 104 :items keyword 63

K

Key to Command command 134, 225 keyboard commands comparison with menu commands 117 finding editor command for 225 in the editor 111 keyboard conventions xii keywords :center 179 :extended-selection 179 :items 63 Kill command 233 kill ring 123–127 copying text from 124–127 marking the region 123 putting text into 124 rotating 126 summary of use 127 killing a process 233

L

lambda list, displaying 133, 222 layouts adding to an interface 175–177 pinboard 214 rearranging 176–177, **209–210** specifying callbacks 183–185 *See also* interfaces layouts, displaying in window browser 269 Link from command 38 linking tools together 37 .lisp files 117 LispWorks clipboard. *See* clipboard LispWorks podium 57–58 Listen command 33, 166 listener basic tutorial 219-221 browsing generic functions from 223 compiling files in 28 current expression, displaying value 222 debugging in the 223 description 217-219 evaluating forms 219-220 execute mode 224 Expression menu. See Expression menu history list 221 History menu 221 loading files in 28 macroexpanding forms 223 next event 224 online help 225 operating on expressions 222 overview 2 pasting selected object into 33 previous event 224 prompt 218 re-evaluating forms 8, 220-221 searching history list 224 *** variable 12 ** variable 12 * variable 12, 17, 166, 220 tracing current expression 222 Values submenu. See Values menu walking forms 223 Listener Bind S command 168 Listener command 217 list-panel class 63 Load command 28, 131, 254, 255, 261 loading files 28 loading tools into the environment 21 local slots, inspecting 158 Long Names button 270 Lower command 210, 271

Μ

Macroexpand command 133 Macroexpand Form command 223 macros defsystem 252 trace 36 major tools, overview 2–3 manipulating values with inspector 163– 166 Manuals command 42 manuals, online. See online help menu commands

check components 203 choosing xi comparison with keyboard commands in editor 117 creating with the interface builder 199-204 debugger commands 223 names, specifying 183 radio components 203 rearranging 209–210 specifying callbacks 185-186 See also interface menu components 202-204 check 203 radio 203 menus creating with the interface builder 180-183, 199-204 rearranging 209-210 See also interface Meta key use of xii methods displaying signatures 153 operating on signatures 153 undefining 80 viewing method combinations 149 Methods menu 79, 148 Trace. See Trace menu Trace submenu. See Trace menu Undefine 80 Modify command 27

Ν

native window representation 271 navigating within files in the editor 118– 119 New command **28**, 117, 189 interface builder 175, 193 new files, creating **28**, 117 Next command command line debugger 224 history list 27 next event repeating **27**, 224

0

Object menu 31 Actions. *See* Actions menu Attributes 210 interface builder 197

Lower 210 Raise 210 objects inspecting 157 operating on 31–34 searching for 26 selecting 26 See also selected object On Symbol command 43 On Tool command 43 online help 41-44 browsing manuals 41 current symbol 43 current tool 43 packages, searching 43 searching 42-43 Open command 28, 108, 117, 120 interface builder 194 opening files 28, 108, 117 operating on objects 31-34 See also objects Options command 23 output compiler 131 editor 115 standard 227-229 output browser 11, 227-229 menu commands. See menu or command name overview 3 overview of major tools 2-3 overview of profiling 238

P

packages current package 30 display of 29-31 in editor 132 searching for documentation 43 Packages button 241, 246, 247 Page Down key 119 Page Up key 119 Parent command 254, 255 Partial Search radio button 43 Paste command 25-26, 123 in class browser 69 in generic function browser 147 in inspector **163**, 165 interface builder 176, 209 pinboard objects moving and resizing 214 podium. See LispWorks podium

Preferences command 30, 50, 66, 69, 79, 103, 115, 153, 243, 268 Preview command 264 previewing a system plan 258-261, 264 Previous command command line debugger 224 history list 27 previous event repeating 27, 224 primary package in editor 132 Print command 28, 118 PRINTER UNIX environment variable 118 printing files 28 process browser 231-234 menu commands. See menu or command name sorting processes 233 processes breaking 234 inspecting 233 killing 233 sorting 232 Processes menu 233 profiler 235-247 choosing packages 241–242 choosing symbols 239-242 description 235-237 example of use 245-247 information returned 238, 243-244 interpreting results 244 menu commands. See menu or command name overview of profiling 238 pitfalls 244 running a profile 243 sorting results 243 specifying code to run 242 symbols that can be profiled 244 prompt in the listener 218 push-button-panel class 65

Q

Quit command 271 quitting the environment 24

R

radio components 203 Raise command 210, 213, 271 readers of a slot, displaying 70 Recompute Events button 260 recursive macroexpansion 223 re-evaluating forms in listener 8, 220-221 Regenerate command 214 repeating commands 26 in the editor 122 repeating the next event 27 repeating the previous event 27 Replace command 26, 128, 129, 256 replacing text 128–129 Report Bug command 101, 224 reporting bugs 97, 101, 224 Restart Frame command 102, 104 restart frames 104 Restarts button 104 Restarts menu 97 Restarts submenu 224 retract callbacks 184 Return From Frame command 102 Reusable checkbox 22 Reusable command 21 Reuse all tools command 21 re-using windows 21 Revert command 118 interface builder 207 reverting a file to the version stored on disk 118 row-layout class 198

S

Save All command 121 interface builder 207 Save As command 118 interface builder 207 Save command 117, 189 interface builder 186, 207 Save Region As command 118 saving all files 121 saving files 117 interface builder 207-208 saving regions of text 118 scrolling text in editor 119 Search command 42 Search Files command 256 searching for objects 26 for text 26, 127-128 history list 224 online manuals 42 Select All command 26, 122, 261 selected object browsing 32 browsing the class of 33

copying 33 displaying documentation 33 finding source code 33 inspecting 33 pasting into listener 33 showing function calls 34 showing generic functions 34, 133, 223 selection callbacks 184 Set command 163, 164 debugger 103 Shift key, use of xii, 111 Short Names button 270 Show Package Names button 30, 105, 167, 269 Signature menu 153 signatures displaying 153 operating on 153 simple-pane-foreground reader 17 Slots menu 31, 69, 82 Copy 166 Inspect 162 Paste 163, 165 Set 163, 164 sort options By Name 36 By Package 36 Unsorted 36 sorting in class browser 66 in editor 116 in inspector 167 in process browser 233 views 35 in window browser 269 source code debugging 93-105 for current expression 222 for object in current frame of debugger 102 for selected object 33 Source command 264 stack frames in the debugger 98 standard action command Browse 32 Class 33 Copy 33 Documentation 33 Find Source 33 Generic Function 34, 133, 223 Inspect 33 Listen 33

standard output 227-229 *standard-output* variable 11 *** variable 12 ** variable 12 * variable 12, 17, 166, 220 Stop command 233 stopping a process 233 swapping editor buffers 120 symbols online help for 43 tracing 132 undefining 133 Symbols button 239, 247 system browsing 254 compiling and loading 255 concatenating 256 creating plans for 260 defining 189, 250-252 executing plans for 260 forcing compilation and loading of members 264 introduction to 249-250 parent system, browsing 254 plan 258 previewing a plan 258–261, 264 ROOT-SYSTEM 254, 257 searching 256 using source files 264 system browser 249-264 Actions area 260 compiling and loading systems 255 creating plans 260 description 252 executing plans 260 File description area 255 Filter area 260 forcing compilation and loading 264 Graph area 254 graph view 253–255 menu commands. See menu or command name output view 261-263 package information 264 parent system, browsing 254 Plan area 260 previewing the plan 258–261 sorting information 263 System area 254 system plan, previewing 258-261, 264 text view 256-258 using 254–255

using source files 264 views available 252 Systems menu 255 Compile **255**, 261 Compile and Load **255**, 261 Concatenate 256 Load **255**, 261 Parent 254, **255** Replace 256 Search Files 256

Т

tab completion 110 in generic function browser 147 text deleting 120 inserting 120 replacing 128–129 saving regions of 118 scrolling in editor 119 searching for 26, 127-128 selecting 26 See also under editor text view in editor 108 in function call browser 139 in system browser 256-258 Toggle Tracing command 37 tools current package of 30 linking together 37 loading into the environment 21 online help for 43 overview of major 2–3 reusing 24 tracing from 36-37 Tools menu 2. 6. 21. 21. 58 Class Browser 60 Clone 22, 23, 37, 121 Editor 21. 108 **Generic Function Browser 145** Inspector 155 Interface Builder 192 Listener 217 Options 23 Output Browser 227 Preferences 30, 50, 66, 69, 79, 103, 115, 153, 243, 268 Profiler 235 Update 22 update 14 Window Browser 265

Trace command 37, 132, 222 Trace Inside command 37 trace macro 36 Trace menu Break on Access 163 Break on Read 163 Break on Write 163 Trace 37 Trace Inside 37 Trace Read 163 Trace with Break 37 Tracing 37 Untrace 37, 163 Untrace All 37 Trace with Break command 37 tracing classes 80 in function call browser 144 in the inspector 163 in inspector 163

U

Undefine command 80, 133, 149 undefining current definition 133 functions 80 generic functions 149 methods 80 Undo command 25 editor 122 UNIX environment variables PRINTER 118 Unsorted option 36 Unstop command 233 unstopping a process 233 Untrace All command 37 Untrace command 37, 163 Update command 14, 22 updating windows 22 using the clipboard 25-26 See also kill ring using the keyboard xii using the mouse ix-xi

V

Value command 133, 222 Value menu Listen 166 values changing in inspector 163–166 Values menu

Class 15, 60, 63 Copy 165 Inspect 23, 157, 160, 162, 164 variables \$ 168 * 12, 17, 166, 220 ** 12 *** 12 *standard-output* 11 Variables menu Actions. See Actions menu Set 103 views in class browser 59 description 34-36 in editor 108 in function call browser 135 in generic function browser 145 graph 45-55, 253-255 hierarchy 65 in inspector 166 output 115, 227-229, 261-263 slots 62-64 sorting items in 35 in system browser 252 text 139, 256-258 Visit Tags File editor command 111

W

Walk command 134 Walk Form command 223 Whole Word radio button 43 window browser 265-271 changing root of graph 267 complete window names, displaying 270 destroying a window 271 different types of window 269 lowering a window 271 menu commands. See menu or command name moving around different windows 271 native representation 271 package information 269 quitting a window 271 raising a window 271 sorting entries 269 using 267 whole environment 271 windows displaying 20 making copies of 22

re-using 21 updating 22 Windows menu 14, 20, 58, 270 Actions. See Actions menu Browse 267 Browse Parent 271 **Browse Representation 271** Browse Screen 271 Destroy 271 Lower 271 Quit 271 Raise 271 in window browser 271 Works menu 58 Exit 6 Object submenu. See Object menu Packages submenu. See Packages menu Symbols submenu. See Symbols menu writers for a slot, displaying 70