# 

Version 4.1



#### **Copyright and Trademarks**

LispWorks for the Windows Operating System Reference Manual

Version 4.1

November 1998

Part number: 3LBDT2A15NC

Copyright © 1987–1998 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 and is subject to change without notice. Harlequin Group plc and its affiliates assume no responsibility or liability for any loss or damage that may arise from the use of any information 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. Harlequin, Delivery, Transducer/PC, The Authoring Book, ClassWorks, and KnowledgeWorks 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."

http://www.harlequin.com/

Europe:	North America:	Asia Pacific:
Harlequin Limited	Harlequin Incorporated	Harlequin Australia Pty. Ltd.
Parrington Uall	One Cambridge Center	Loval 19

Level 12 12 Moore Street Canberra, ACT 2601 Australia

telephone +61 2 6206 5522 fax +61 2 6206 5525

Barrington Hall Barrington Cambridge CB2 5RG UK

telephone +44 1223 873 800 fax +44 1223 873 873

#### One Cambridge Center Cambridge, MA 02142 USA

telephone +1 617 374 2400 fax +1 617 252 6505

# Contents

## Preface v

- 1 The CLOS Package 1
- 2 The COLOR Package 9
- 3 The COMM package 29
- 4 The COMMON-LISP Package 35
- 5 The COMPILER Package 73
- 6 The DBG Package 77
- 7 The DSPEC Package 87
- 8 The EXTERNAL-FORMAT Package 89
- 9 The HCL Package 95
- 10 The LINK-LOAD Package 149
- 11 The LISPWORKS Package 159
- 12 The MP Package 217
- 13 The PARSERGEN Package 263

- 14 The SCM Package 265
- 15 The SQL Package 269
- 16 The STREAM package 321
- 17 The SYSTEM Package 343
- The WIN32 package (including DDE) 363
   General WIN32 symbols 363
   DDE client interface reference entries 364
   DDE server interface reference entries 393

Index 405

# Preface

## About this manual

The *LispWorks Reference Manual* is organized by package. Each chapter contains detailed reference material for all the exported symbols in a given package. Within any chapter, the symbols are organized alphabetically (ignoring non-alphanumeric characters that are common in Lisp symbols, such as \*). The chapters themselves are organized alphabetically, as follows:

- Chapter 1, "The CLOS Package", describes Harlequin's extensions to CLOS, the Common Lisp Object System.
- Chapter 2, "The COLOR Package", describes symbols available in the COLOR package. Symbols from this package can be used to control the colors used in Common Lisp applications.
- Chapter 3, "The COMM package", describes the functions providing the TCP/IP interface.
- Chapter 4, "The COMMON-LISP Package", describes Harlequin's extensions to symbols in the COMMON-LISP package. You should refer to the Common Lisp Hyperspec, supplied in HTML format with Lisp-Works, for full documentation about standard Common Lisp symbols.
- Chapter 5, "The COMPILER Package", describes symbols available in the COMFILER package.

- Chapter 6, "The DBG Package", describes symbols available in the DBG package, used to configure the debugging information produced by LispWorks.
- Chapter 7, "The DSPEC Package", describes the symbols available in the DSPEC package, which are used for definition specifications.
- Chapter 8, "The EXTERNAL-FORMAT Package", describes symbols available in the EXTERNAL-FORMAT package.
- Chapter 9, "The HCL Package", describes symbols available in the HCL package.
- Chapter 10, "The LINK-LOAD Package", describes the symbols available in the LINK-LOAD package.
- Chapter 11, "The LISPWORKS Package", describes symbols available in the LISPWORKS package.
- Chapter 12, "The MP Package", describes symbols available in the MP package, giving you access to the multi-processing capabilities of LispWorks.
- Chapter 13, "The PARSERGEN Package", describes symbols available in the **PARSERGEN** package, the LispWorks parser generator.
- Chapter 14, "The SCM Package", describes symbols available in the scm package that control aspects of source control management.
- Chapter 15, "The SQL Package" documents symbols used in accessing LispWorks ODBC and SQL functionality.
- Chapter 16, "The STREAM package" documents the symbols available in the **STREAM** package that provide users with the functionality to define their own streams for use by the standard I/O functions.
- Chapter 17, "The SYSTEM Package", describes symbols available in the **SYSTEM** package.
- Chapter 18, "The WIN32 package (including DDE)", describes symbols available in the win32 package.

Many chapters in the *LispWorks Reference Manual* should be used in conjunction with relevant chapters in the *LispWorks User Guide* that describe use of the module pertaining to the package.

Reference material for some aspects of LispWorks can be found in places other than this manual.

- As mentioned above, refer to the Common Lisp Hyperspec for documentation about Common Lisp itself.
- Refer to the *CAPI Reference Manual* for details about symbols in the CAPI package, and related packages such as the GP package.
- Refer to the *LispWorks Foreign Language Interface User Guide and Reference Manual* for details on integrating C and Common Lisp source code.
- Refer to the *LispWorks Editor User Guide* for documentation on the Lisp-Works Editor.

# The CLOS Package

This chapter describes Harlequin's extensions to CLOS, the Common Lisp Object System.

## break-new-instances-on-access

Summary	Traps new instances of a given class, based on access modes. Note that this function is deprecated.	
Package	clos	
Signature	break-new-inst	ances-on-access class-designator &key read write slot-names => t
Arguments	class-designator	The class to trap.
	read	If $t$ , trap read accesses to the class. The default is $t$ .
	write	If $t$ , trap write accesses to the class. The default is $t$ .

## The CLOS Package

	slot-names	t means break on all slots, otherwise this should be a list of symbols which are the slot-names on which to break. The default is t.
Values	Returns t.	
Description	be trapped for t <i>names</i> . Note tha	tances of the class given by <i>class-designator</i> to he access modes given by <i>read</i> , <i>write</i> and <i>slot-</i> t this function is deprecated. You should now instances-on-access instead.
See also	trace-new-inst	tances-on-access

## break-on-access

Summary	Breaks to the debugger when an instance of a class is accessed. Note that this function is deprecated.	
Package	clos	
Signature	break-on-access instance &key read write slot-names => t	
Arguments	The keyword ar break:	guments control which type of access cause a
	read	Non-nil means break on reading a slot. The default is t.
	write	Non-nil means break on writing a slot. The default is t.
	slot-names	t means break on all slots, otherwise this should be a list of symbols which are the slot-names on which to break. The default is t.

A common use of this function is to find where a slot is being changed in a complex program.

#### Values Returns t.

Description A useful debugging function which causes access to *instance* to break to the debugger. Accesses include calls to slotnames and accessor function defined by the class of *instance*. Other instances of the same class are unaffected and your can remove the break by calling clos:unbreak-on-access. Breaks to the debugger when an instance of a class is accessed. Note that this function is deprecated. You should now use trace-on-access instead.

See also trace-on-access

## trace-new-instances-on-access

Summary	Traces new instances of a given class, based on access modes.	
Package	clos	
Signature	break-new-inst	ances-on-access class-designator &key read write slot-names break => t
Arguments	class-designator	The class to trap.
	read	If $t$ , trap read accesses to the class. The default is $t$ .
	write	If $t$ , trap write accesses to the class. The default is $t$ .
	slot-names	τ means trace on all slots, otherwise this should be a list of symbols which are the slot-names to trace. The default is τ.

	break	When non-nil, the debugger is entered. When nil, messages are printed to *trace- output*.
Values	Returns t.	
Description		tances of the class given by <i>class-designator</i> to e access modes given by <i>read</i> , <i>write</i> and <i>slot-</i>
		vhen used with the break keyword, replaces function break-new-instances-on-access.
Example	(break-new-ins	tances-on-access 'capi:display-pane :slot-names nil)
See also		tances-on-access nstances-on-access ss

## trace-on-access

Summary	Invokes the trac accessed.	ce facilities when an instance of a class is
Package	clos	
Signature	trace-on-acces	ss instance &key read write slot-names break => t
Arguments	The keyword arguments control which type of access cause a break:	
	read	Non-nil means break on reading a slot. The default is t.
	write	Non-nil means break on writing a slot. The default is t.

	slot-names	t means break on all slots, otherwise this should be a list of symbols which are the slot-names on which to break. The default is t.
	break	When non-nil, the debugger is entered. When nil, messages are printed to *trace- output*.
		of this function is to find where a slot is being omplex program.
Values	Returns t.	
Description	A useful debugging function which causes access to <i>instance</i> to invoke the trace facilities. Accesses include calls to <i>slot-names</i> and accessor function defined by the class of <i>instance</i> . Other instances of the same class are unaffected and your can remove the break by calling close:untrace-on-access. Breaks to the debugger when an instance of a class is accessed.	
		when used with the break keyword, replaces function break-on-access.
See also	untrace-on-ac trace-new-ins break-on-acce	tances-on-access

## unbreak-new-instances-on-access

Summary	Removes the trapping installed by break-new-instances- on-access. Note that this function is deprecated.
Package	clos
Signature	unbreak-new-instances-on-access <i>class-designator</i> => t

#### The CLOS Package

Arguments	<i>class-designator</i> The class whose trap you want to remove.
Values	Returns t.
Description	Removes the trapping installed by break-new-instances- on-access. Note that this function is deprecated. You should now use untrace-new-instances-on-access instead.
See also	untrace-new-instances-on-access

## unbreak-on-access

## Function

Summary	Removes the trapping installed by break-on-access. Note that this function is deprecated.
Package	clos
Signature	unbreak-on-access instance
Arguments	<i>instance</i> A class instance
Values	Returns t.
Description	Removes any break installed on <i>instance</i> by clos:break-on- access. See untrace-on-access for details. Note that this function is deprecated. You should now use untrace-on- access instead.
See also	untrace-on-access

## untrace-new-instances-on-access

Function

Summary Removes the trapping installed by trace-new-instanceson-access.

Package	clos	
Signature	unbreak-new-instances-on-access class-designator break => t	
Arguments	class-designator	The class whose trap you want to remove.
	break	When non-nil, the debugger is entered. When nil, messages are printed to *trace- output*.
Values	Returns t.	
Description	Removes the trapping installed by trace-new-instances- on-access.	
See also	trace-new-instances-on-access untrace-on-access	

## untrace-on-access

Summary	Removes the trapping installed by trace-on-access.	
Package	clos	
Signature	unbreak-on-acc	ess instance break => t
Arguments	instance	A class instance
	break	When non-nil, the debugger is entered. When nil, messages are printed to *trace- output*.
Values	Returns t.	
Description	Removes any tr access.	race installed on <i>instance</i> by clos:break-on-

## The CLOS Package

See also trace-on-access untrace-new-instances-on-access

# The COLOR Package

This chapter describes symbols available in the COLOR package. Symbols from this package can be used to control the colors used in Common Lisp applications. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

## apropos-color-alias-names

Summary	Returns color al	liases containing a given string.
Package	color	
Signature	apropos-color-	alias-names <i>substring</i> => <i>list</i>
Arguments	substring	A string.
Values	list	A list of symbols.
Description	string and whic	f symbols whose symbol-names contain <i>sub</i> - h are defined as aliases in the color-database liases. By convention these are in the key-

Example	In this example, a color alias is defined for the color indianred1. apropos-color-alias-names only returns this alias, rather than both the alias and the original color, despite the similarity in the names.
	COLOR 8 > (define-color-alias :myindianred1 :indianred1)
	(#S(COLOR-ALIAS COLOR :INDIANRED1))
	COLOR 9 > (apropos-color-names "INDIANRED1") (:INDIANRED1 :MYINDIANRED1)
	COLOR 10 > (apropos-color-alias-names "INDIANRED1") (:MYINDIANRED1)
	COLOR 11 >
See also	apropos-color-names apropos-color-spec-names get-all-color-names

# apropos-color-names

Summary	Returns colors a	nd color aliases containing a given string.
Package	color	
Signature	apropos-color-	names substring => list
Arguments	substring	A string.
Values	list	A list of symbols.
Description	string and which	symbols whose symbol-names contain <i>sub</i> - h are present in the color-database defining convention these are in the keyword pack-

Example	COLOR-4> (color:apropos-color-names "RED")
	(:ORANGERED3 :ORANGERED1 :INDIANRED3 :INDIANRED1
	:PALEVIOLETRED :RED :INDIANRED :INDIANRED2
	:INDIANRED4 :ORANGERED :MEDIUMVIOLETRED
	:VIOLETRED :ORANGERED2 :ORANGERED4 :RED1 :RED2 :RED3
	:RED4 :PALEVIOLETRED1 :PALEVIOLETRED2 :PALEVIOLETRED3
	:PALEVIOLETRED4 :VIOLETRED3 :VIOLETRED1 :VIOLETRED2
	:VIOLETRED4)

See also apropos-color-alias-names apropos-color-spec-names get-all-color-names

## apropos-color-spec-names

Summary	Returns colors	containing a given string.
Package	color	
Signature	apropos-color-spec-names substring => list	
Arguments	substring	A string.
Values	list	A list of symbols.
Description	string and whic	f symbols whose symbol-names contain <i>sub</i> - th are defined as original entries in the color- ng color aliases. By convention these are in ackage.
Example	•	efine-color-alias :mygray100 :gray100) AS COLOR :GRAY100))
	COLOR 15 > (ap (:MYGRAY100 :G	propos-color-names "GRAY100") GRAY100)
	COLOR 16 > (ap (:GRAY100)	<pre>&gt;]ropos-color-spec-names "GRAY100")</pre>
	COLOR 17 >	

See also apropos-color-alias-names apropos-color-names get-all-color-names

## color-<component>

Summary	Returns the associated component of a color specification.
Package	color
Signature	<pre>color-red color-spec =&gt; color-component color-green color-spec =&gt; color-component color-blue color-spec =&gt; color-component color-hue color-spec =&gt; color-component color-saturation color-spec =&gt; color-component color-value color-spec =&gt; color-component</pre>
Arguments	<i>color-spec</i> A color specification.
Values	<i>color-component</i> A color component from the appropriate color model.
Description	If <i>color-spec</i> is not from the appropriate color model (:rgb in the case of color-red, color-green and color-blue, and :hsv in the case of color-hue, color-saturation and color-value) then the component is calculated.
Description Example	the case of color-red, color-green and color-blue, and :hsv in the case of color-hue, color-saturation and color-
	the case of color-red, color-green and color-blue, and :hsv in the case of color-hue, color-saturation and color- value) then the component is calculated. COLOR 31 > (color:make-rgb 1.0s0 0.0s0 0.0s0)
	<pre>the case of color-red, color-green and color-blue, and :hsv in the case of color-hue, color-saturation and color- value) then the component is calculated. COLOR 31 &gt; (color:make-rgb 1.0s0 0.0s0 0.0s0) #(:RGB 1.0s0 0.0s0 0.0s0) COLOR 32 &gt; (color-red *)</pre>
	<pre>the case of color-red, color-green and color-blue, and :hsv in the case of color-hue, color-saturation and color- value) then the component is calculated. COLOR 31 &gt; (color:make-rgb 1.0s0 0.0s0 0.0s0) #(:RGB 1.0s0 0.0s0 0.0s0) COLOR 32 &gt; (color-red *) 1.0s0 COLOR 33 &gt; (color-green **)</pre>

See also	make-hsv
	make-rgb
	make-gray
	color-model
	color-level

## \*color-database\*

## Variable

Summary	The current color-database.
Package	color
Description	This should contain definitions for all the colors used in the environment when you start it. Those colors are determinable from the file config/r-color.lisp file and from your ownu-color.lisp file.
Example	To replace the current color database with a new one, do the following: (setf_color:*color-database* (color:make-color-db))
See also	delete-color-translation read-color-db load-color-database

color-level			Function
Summary	Returns the gra	y level of a color specification.	
Package	color		
Signature	color-level co	lor-spec => gray-level	
Arguments	color-spec	A color specification.	
Values	gray-level	Color component from the :gray	model.

## The COLOR Package

Description	Return the gray level of <i>color-spec</i> . If <i>color-spec</i> is not from the <b>:GRAY</b> model, the component is calculated.
Example	COLOR 2 > (color:make-gray 0.666667s0) #(:GRAY 0.66667s0)
	COLOR 3 > (color-level *) 0.66667S0
	COLOR 4 >
See also	make-hsv make-rgb make-gray color-model color- <component></component>

## color-model

Summary	Returns the color-model for a color-spec.
Package	color
Signature	color-model color-spec => color-model
Arguments	<i>color-spec</i> A color specification.
Values	color-model :gray, :rgb, 0r :hsv.
Example	COLOR 29 > (color:make-gray 0.666667s0) #(:GRAY 0.66667s0)
	COLOR 30 > (color-model *) :GRAY
	COLOR 31 >
See also	<pre>make-hsv make-rgb make-gray color-<component> color-level</component></pre>

## colors=

## Function

Summary	Tests to see if two colors are equal.	
Package	color	
Signature	colors= color1 color2 & optional tolerance => bool	
Arguments	color1	A color specification.
	color2	A color specification.
	tolerance	A tolerance level within which <i>color1</i> and <i>color2</i> may vary. The default value is 0.001s0.
Values	bool	t if the two colors are equal within the given tolerance, הוו otherwise.
Description	Return t if the t	wo colors are equal to the given tolerance.
See also	ensure- <command/> convert-color	

## convert-color

Summary	Return the repr graphics port.	esentation of a color specification on a given
Package	color	
Signature	convert-color	port color &key errorp => color-rep
Arguments	port	A graphics port.
	color	A color specification.
	errorp	If $t$ , check for errors. By default, this is $t$ .

Values	<i>color-rep</i> Representation of <i>color</i> on <i>port</i> .
Description	Return the representation of <i>color</i> on the given graphics port <i>port</i> . In CLX, this is the "pixel" value, which corresponds to an index into the default colormap. It is more efficient to use the result of convert-color in place of its argument in draw- ing function calls, but the penalty is the risk of erroneous col- ors being displayed should the colormap or the colormap entry be changed.
See also	colors= ensure- <command/>

## define-color-alias

Summary	Lets you define an alias for a color specification or alias.	
Package	color	
Signature	define-color-a	lias name color &optional if-exists => alias
Arguments	name	The name of the new alias.
	color	A color specification for the new alias.
	if-exists	This can be one of the following:
		:replace — Replace any existing alias.
		<pre>:error — Raise an error if alias is already defined.</pre>
		: ignore — Ignore redefinition of an alias.
		By default, it is :replace.
Values	alias	The color alias.
Description	Define <i>name</i> to be a color alias for <i>color</i> , which may be another color alias or a color-spec.	

Example 1 COLOR 16 > (define-color-alias :mygray :darkslategray) (#S(COLOR-ALIAS COLOR :DARKSLATEGRAY)) COLOR 17 > (define-color-alias :mygray :darkslategray :error) Error: :MYGRAY names an existing alias for #(:RGB 0.184313350 0.30980350 0.30980350) 1 (continue) Replace :MYGRAY with the alias :DARKSLATEGRAY 2 Continue, without redefining alias :MYGRAY 3 Try a new name for the alias, instead of :MYGRAY 4 (abort) Return to level 0. 5 Return to top loop level 0. 6 Destroy process. Type :c followed by a number to proceed or type :? for other options COLOR 18 : 1 > Example 2 COLOR 19 > (define-color-alias :lispworks-blue (make-rgb 0.70s0 0.90s0 0.99s0)) (#S(COLOR-ALIAS COLOR #(:RGB 0.69999950 0.950 0.9950))) COLOR 20 > See also get-color-alias-translation get-color-spec

## define-color-models

Macro

Summary	Defines <i>all</i> the c	olor models.
Package	color	
Signature	define-color-mo	odels model-descriptors=> color-models
Arguments	model-descriptors	s A list, each element being a model-descrip- tor.
Values	color-models	The color models defined.

Description A model descriptor has the syntax: (model-name component-descr\*) A *component-descr* is a list: (component-name lowest-value highest-value) The default color models are defined by the following form: (define-color-models ((:rgb (red 0.0 1.0) (green 0.0 1.0) (blue 0.0 1.0)) (:hsv (hue 0.0 5.99999) (saturation 0.0 1.0) (value 0.0 1.0)) (:gray (level 0.0 1.0)))) If you want to keep existing color models, add your new ones to this list: only one define-color-models form is recognized. The form should be compiled. Example To replace the HSV color model with a CMYK model, while retaining the other color models: (define-color-models ((:rgb (red 0.0 1.0)

```
(green 0.0 1.0)
(blue 0.0 1.0)
(:cmyk (cyan 0.0 1.0)
(magenta 0.0 1.0)
(yellow 0.0 1.0)
(black 0.0 1.0)
(:gray (level 0.0 1.0))))
```

## delete-color-translation

Summary	Removes an en	try from the color-database.
Package	color	
Signature	delete-color-	translation <i>color-name</i> => <no values=""></no>
Arguments	color-name	A defined color spec or alias.

Values	None.
Description	Both original entries and aliases can be removed.
See also	load-color-database *color-database* read-color-db

## ensure-<command>

Summary	Return a color specification for a given model. The model depends on the particular function called	
Package	color	
Signature	<pre>ensure-rgb color-spec =&gt; color-spec ensure-hsv color-spec =&gt; color-spec ensure-gray color-spec =&gt; color-spec ensure-model-color color-spec model =&gt; color-spec ensure-color color-spec match-color-spec =&gt; color-spec</pre>	
Arguments	For all functions:	
	<i>color-spec</i> A color specification.	
	match-color-spec A color specification.	
	model A color-model (:rgb, :hsv or :gray).	
Values	<i>color-spec</i> A color specification.	
Description	These functions all return a color-spec, given (at least) a color-spec as argument.	
	ensure-rgb, ensure-hsv and ensure-gray all return a color- spec in the appropriate model. If the color-spec given as an argument is in the same model, it is just returned. Otherwise a new color-spec for that model is calculated. Thus, ensure- rgb returns a color-spec in the RGB color-model, whatever color-model is used in the color-spec given as an argument.	

ensure-model-color is similar to the above three functions, except that a color-model *model* is explicitly passed as an argument to the function. The color-spec returned is in the color-model specified by *model*.

ensure-color returns a color-spec for *color-spec*, in the colormodel specified by *match-color-spec*. Thus, color-specs may be converted from one model to another with having to explicitly state the color-model.

- Example 1 COLOR 36 > (ensure-hsv (color:make-rgb 0.70s0 0.90s0 0.99s0)) #(:HSV 4.31033S0 0.707069S0 0.99S0) COLOR 37 > (ensure-gray (color:make-rgb 0.70s0 0.90s0 0.99s0)) #(:GRAY 0.863331S0)
- Example 2 COLOR 39 > (ensure-model-color (color:make-rgb 0.70s0 0.90s0 0.99s0) :hsv) #(:HSV 4.31033s0 0.707069s0 0.99s0)
- Example 3 COLOR 43 > (ensure-color (make-hsv 0.70s0 0.90s0 0.99s0) (make-rgb 0.70s0 0.90s0 0.99s0)) #(:RGB 0.99s0 0.890999s0 0.92069924)
- See also convert-color colors=

## get-all-color-names

Summary	Returns a list of	all color-names in the color database.
Package	color	
Signature	get-all-color-r	names &optional sort => color-names
Arguments	sort	If t, sort list of color names alphanumeri- cally. By default, this is nil.

Values	color-names	A list of all color names in the color data- base.
Description	vention these an returned list is a	Call color-names in the color database. By con- re symbols in the keyword package. The alphanumerically sorted on the symbol- tional argument is non-nil.
See also	apropos-color apropos-color apropos-color	-spec-names

# get-color-alias-translation

Summary	Return the ultim	nate color name associated with <i>color-alias</i> .
Package	color	
Signature	get-color-alias	s-translation <i>color-alias</i> => color-name
Arguments	color-alias	A defined color alias.
Values	color-name	The color name associated with <i>color-alias</i> .
Example		lor:define-color-alias :lispworks-blue (color:make-rgb 0.70s0 0.90s0 0.99s0)) S COLOR #(:RGB 0.699999S0 0.9S0 0.99S0)))
		lor:define-color-alias :color-background :lispworks-blue) 5 COLOR :LISPWORKS-BLUE))
	COLOR 25 > (co]	lor:define-color-alias :listener-background :color-background)
	(#S(COLOR-ALIAS	S COLOR : COLOR-BACKGROUND))
	COLOR 26 > (get	t-color-alias-translation :listener-background)
	:LISPWORKS-BLUE	2

## The COLOR Package

	COLOR 27 > (color:get-color-alias-translation :color-background)
	:LISPWORKS-BLUE
	COLOR 28 >
See also	define-color-alias get-color-spec

# get-color-spec

Summary	Returns the cole	or-spec for a color.
Package	color	
Signature	get-color-spec	color => color-spec
Arguments	color	A defined color specification, color alias, or an original color name.
Values	color-spec	A color specification.
Description		or-spec for <i>color</i> , which can be a color-spec, a n original color name.
Example	-	olor:define-color-alias :lispworks-blue (color:make-rgb 0.70s0 0.90s0 0.99s0)) & COLOR #(:RGB 0.699999s0 0.9s0 0.99s0)))
		olor:define-color-alias :color-background :lispworks-blue) & COLOR :LISPWORKS-BLUE))
		olor:define-color-alias :listener-background :color-background) NS COLOR :COLOR-BACKGROUND))
		et-color-spec :listener-background) 1950 0.950 0.9950)
	-	et-color-spec :color-background) 1950 0.950 0.9950)

	COLOR 33 > (get-color-spec :lispworks-blu #(:RGB 0.699999S0 0.9S0 0.99S0)		
	COLOR 34 > (get-color-spec #(:RGB 0.70s0 0.90s0 0.99s0))		
	#(:RGB 0.699999S0 0.9S0 0.99S0)		
	COLOR 35 >		
See also	define-color-alias get-color-alias-translation		

## load-color-database

Summary	Loads a color database.
Package	color
Signature	load-color-database <i>data</i> => <no values=""></no>
Arguments	<i>data</i> A description of a color database.
Values	None.
Description See also	This loads the color database with color definitions contained in <i>data</i> , which should have been obtained via the functions color:read-color-db. The colors thus defined may not be replaced by color aliases. *color-database*
	delete-color-translation read-color-db
make-gray	Function
Summary	Returns a color specification in the gray model.
Package	color

## The COLOR Package

Signature	make-gray level	=> color-spec	
Arguments	level	A color component used to define the gray level required.	
Values	color-spec	A color specification.	
Description	Return a color-spec in the :GRAY model with component level.		
	in the most effic	floats are used for the component; this results cient color conversion process. However, any umber type can be used.	
Example	COLOR 25 > (cc #(:GRAY 0.6666 COLOR 26 >	blor:make-gray 0.66667s0) 7s0)	
See also	make-hsv make-rgb color-model color- <compon color-level</compon 	ent>	

## make-hsv

Summary	Returns a color model.	specification in the hue-saturation-value
Package	color	
Signature	<pre>make-hsv hue saturation value =&gt; color-spec</pre>	
Arguments	hue	A hue component.
	saturation	A saturation component.
	value	A value component.
Values	color-spec	A color specification.

Description	Return a color-spec in the <b>:</b> HSV model with components <i>hue saturation</i> and <i>value</i> .		
	Note that short-floats are used for each component; this results in the most efficient color conversion process. How- ever, any floating-point number type can be used.		
Example	COLOR 27 > (color:make-hsv 1.2s0 0.5s0 0.9s0) #(:HSV 1.2S0 0.5S0 0.9S0)		
	COLOR 26 >		
See also	<pre>make-rgb make-gray color-model color-<component> color-level</component></pre>		

#### Function make-rgb Summary Returns a color specification in the red-green-blue model. Package color Signature make-rgb red green blue => color-spec Arguments red A red component. A green component. green blue A blue component. Values color-spec A color specification. Description Return a color-spec in the :RGB model with components red,

*green* and *blue*. Note that short floats are used for each component; this results in the most efficient color conversion process. However, any floating point number type can be used.

#### The COLOR Package

Example The object returned by the following call defines the color red in the RGB model: COLOR 25 > (color:make-rgb 1.0s0 0.0s0 0.0s0) #(:RGB 1.0s0 0.0s0 0.0s0) COLOR 26 > See also make-hsv make-gray color-model color-<component> color-level

## read-color-db

Summary	Reads the color definitions contained in a file.		
Package	color		
Signature	read-color-db	<pre>&amp;optional file =&gt; color-databas</pre>	e
Arguments	file	A filename or pathname cont color definitions to be read. It given, read-color-db uses th definitions file in the LispWo	f <i>file</i> is not ne default color
Values	color-database	A database definition.	
Description	This reads color definitions from the given <i>file</i> (a filename or pathname). The returned data structure can be passed to color:load-color-database. The format of the file is: #(:RGB 1.0s0 0.980391s0 0.980391s0) snow #(:RGB 0.972548s0 0.972548s0 1.0s0) GhostWhite 		
		ins a color definition which con e. The names are converted to	

interned in the keyword package. Whitespace in names is preserved.

See also load-color-database \*color-database\* delete-color-translation The COLOR Package

# The COMM package

The comm package provides the TCP/IP interface. TCP/IP sockets can be used to communicate between processes and machines. The TCP/IP mechanism allows LispWorks to connect to or implement a server. Before the interface can be used the module "comm" must be loaded using

(require "comm")

This chapter provides reference entries for the functions in the COMM package.

# open-tcp-stream

Summary	1	nnect to a socket on another machine and n object for the connection.
Package	comm	
Signature	open-tcp-strea buffered => stream	m hostname service &key direction element-type m-object
Arguments	hostname	An integer or string.
	service	A string or a fixnum.

	direction	One of :input, :output OF :io.
	element-type	One of base-char, unsigned-byte or signed-byte.
	errorp	A boolean.
Values	stream-object	A socket stream.
Description	The open-tcp-stream function attempts to connect to a socket on another machine and returns <i>stream-object</i> for the connection if successful. The server machine to connect to is given by <i>hostname</i> , which can be one of the following:	
	• A string nam	ning the host, for example ce.com"
	• A string pro	viding the IP address, for example 7.75"
	<ul> <li>An integer IP address in network order, for example #xCC47B14B</li> <li>The name of the service to provide is given by <i>service</i>. If <i>vice</i> is a string, the location of the file specifying the name the services available varies, but typically on Windows 9 is called services and is stored in the windows directory., on Windows NT it is the file</li> </ul>	
	%SystemRoot%\s	ystem32\drivers\etc\SERVICES
	The <i>service</i> can a of the desired co	lso be a fixnum representing the port number onnection.
	default value is	the connection is given by <i>connection</i> . Its :io. The element type of the connection is <i>t-type</i> , and is base-char by default.
	If <i>errorp</i> is nil, f error is signaled	ailure to connect returns נות, otherwise an
Example		xample opens an HTTP connection to a given ves the root page:

(with-open-stream (http (comm:open-tcp-stream "webhost" 80)) (format http "GET / HTTP/1.0~C~C~C" (code-char 13) (code-char 10) (code-char 13) (code-char 10)) (force-output http) (write-string "Waiting to reply...") (loop for ch = (read-char-no-hang http nil :eof) until ch do (write-char #).) (sleep 0.25) finally (unless (eq ch :eof) (unread-char ch http))) (terpri) (loop for line = (read-line http nil nil) while line do (write-line line))) See also start-up-server

## start-up-server

Summary Starts a TCP server.

ounnurj		
Package	comm	
Signature	start-up-serve	er &key function announce service process-name =>
Arguments	<i>function</i> A function name.	
	announce	A keyword.
	service	An integer or string.
	process-name	A symbol or expression.
Values	None.	
Description	The start-up-server function starts a TCP server. Use mp:process-kill to kill the server, and comm:open-tcp-stream to send messages from another client to the server.	

The *function* argument provides the name of the function that processes connections. When a connection is made *function* is called with the connected socket handle, at which point you can make a stream using make-instance and communicate with the client. The server does not accept more connections until *function* returns, so normally it creates another lightweight process to handle the connection. If no function is specified the built-in Lisp listener server is used. See the examples section below.

The *announce* keyword causes a message to appear when the server is started. The value of this keyword is the stream to which the startup message is displayed.

The *service* argument specifies the name of the service. The location of the file specifying the names of services available varies, but typically on Windows 95 it is called **SERVICES** and is stored in the **Windows** directory., and on Windows NT it is the file

%SystemRoot%\system32\drivers\etc\SERVICES

The default value for *service* is lispworks.

The *process-name* specifies the process name. The default is constructed from the service name in the following fashion:

(format nil "~S server" service)

Examples The following example uses the built-in Lisp listener server:

(comm:start-up-server :service 10243)

It makes a Lisp listener server on port 10243 (check with local network managers that this port number is safe to use). When a client connects to this, Lisp calls **READ**. The client should send a string using CL syntax followed by a newline. This string is used to name a new light-weight process that runs a Lisp listener. When this has been created, the server waits for more connections.

The next example illustrates the use of the *function* argument. For each line of input read by the server it writes the line back with a message. The stream generates **EOF** if the other end closes the connection.

```
(defvar *talk-port* 10244) ; a free TCP port number
(defun make-stream-and-talk (handle)
 (let ((stream (make-instance 'comm:socket-stream
                               :socket handle
                               :direction :io
                               :element-type
                                  'base-char)))
 (mp:process-run-function (format nil "talk ~D"
                                   handle)
                           ()
                           'talk-on-stream stream)))
(defun talk-on-stream (stream)
  (unwind-protect
      (loop for line = (read-line stream nil nil)
            while line
            do
            (format stream "You sent: '~A'~%" line)
            (force-output stream))
      (close stream)))
(comm:start-up-server :function 'make-stream-and-talk
                      :service *talk-port*)
```

See also open-tcp-stream socket-stream

# socket-stream

Class

Summary The socket stream class.

Superclasses stream

Initargs:socketA socket handle.:directionOne of :input, :output, or :io.

### The COMM package

element-type An element type.

ACCESSORS socket-stream-socket

Description The socket-stream class implements a buffered stream connected to a socket. The socket direction, specified by :socket, and the direction, specified by :direction, must be passed for a meaningful stream to be constructed. Common Lisp input functions such as read-char will see end-of-file if the other end of the socket is closed. The :element-type keyword specifies the expected element type of the stream traffic.

Example The following makes a bidirectional stream connected to a socket specified by handle.

(make-instance 'comm:socket-stream :socket handle :direction :io :element-type 'base-char)

See also open-tcp-stream start-up-server

This chapter describes Harlequin's extensions to symbols in the COMMON-LISP package, which is used by default. This chapter notes only those differences between Harlequin Common Lisp and the ANSI standard. You should refer to the Common Lisp Hyperspec, supplied in HTML format with LispWorks, for full documentation about standard Common Lisp symbols.

# base-string

Summary	The base string	type.
Package	common-lisp	
Signature	base-string len	gth
Arguments	length	The length of the string (or *, meaning any).
Description	The type of base strings.	

*Type* 

character	Type
Summary	The character type or repertoire.
Package	common-lisp
Signature	character repertoire
Description	The type of characters (when <i>repertoire</i> is <b>*</b> or not given), or one of the character repertoires:
	:ENGLISH :STANDARD :ASCII :BASE = :LATIN-1 = :SBCS :KANJI :KATAKANA :HIRAGANA :DBCS

# characterp

Summary	Tests is an object is a character or character repertoire.	
Package	common-lisp	
Signature	characterp object & optional repertoire => bool	
Arguments	object	The object to be tested.
	repertoire	A character repertoire.
Values	bool	t if <i>object</i> is a character (or character reper- toire, if <i>repertoire</i> is given); nil otherwise.
Description	This is the predicate for characters (when repertoire is not given), or character repertoires.	
See also	character	

Example	CL-USER 234 > T	(characterp #\a :ascii)	
close		Generic Func	ction
Summary	Redefines the s generic functio	tandard close function, implementing it a n.	as a
Package	common-lisp		
Signature	close <i>stream</i> &	xey abort => result	
Arguments	stream	A stream.	
	abort	A generalized boolean.	
Values	result	A result value.	
Description	default method stream class se	Lose is implemented as a generic function. I for close provided by the fundamental- ts a flag for open-stream-p. The result val per the Common Lisp ANSI specification.	ue
See also	fundamental-s	tream	
coerce		Func	ction
Summary		tandard coerce function, allowing it to tak Lisp type specifier.	ke
Package	common-lisp		
Signature	coerce object n	esult-type => result	
Arguments	object	A Lisp object.	

	result-type	A type specifier.
Values	result	An object of type <i>result-type</i>
Description	required by the allowed for coe	nction still performs those conversions e standard, but a larger set of type specifiers is ercion. A type-error is signalled if <i>result</i> can- l as the <i>result-type</i> specifies.
See also	concatenate	
compile		Function
Summary	Compile a lam	bda expression into a code vector.
Package	common-lisp	
Signature	compile name &	coptional definition => name, function
Arguments	definition	If supplied, this is a lambda-expression to be compiled.
		If not supplied, then the lambda-expression used is the current definition of the name (in this case <i>name</i> must be a non-nil symbol with an uncompiled definition).
	name	If not nil, this is the symbol that is to receive the compiled function as its global function definition.
Values	or when <i>name</i> i Such compiled	is returned, being the <i>name</i> symbol if supplied, s nil the compiled function definition itself. -function objects are not printable (but see other than as # <compiled for="" function="" sym-<="" td=""></compiled>

- Description compile calls the compiler to translate a lambda expression into a code vector containing an equivalent sequence of host specific machine code. A compiled function typically runs between 10 and 100 times faster. It is generally worth compiling the most frequently called Lisp functions in a large application during the development phase. The compiler detects a large number of programming errors, and the resulting code runs sufficiently faster to justify the compilation time, even during development.
- Examples (defun fn (...) ...) ; interpreted definition for fn
   (compile 'fn ; replace with compiled
   ; definition
   (compile nil '(lambda (x) (\* x x)))
   ; returns compiled squaring function
   (compile 'cube '(lambda (x) (\* x x x)))
   ; defun and compile in one
- Notes See declare for a list of the declarations that alter the behavior of the compiler.
- See also compile-file disassemble declare

# compile-file

### Function

	output-file	Specifies the location of the output file. This keyword is useful if you are using a differ- ent file-extension for binary files. If you use different file-extensions for binary files, you must inform LispWorks of this by pushing the file-extension string onto the variable sys::*binary-file-types*. If you fail to do this, LispWorks assumes that these files are text rather than compiled files.See the exam- ple below.
	:print	This argument controls the printing of infor- mation about the compilation. It can have the following values:
		nil— No information is printed.
		<b>2</b> — Full information is printed out.
		<ul> <li>The information printed out consists of all warning messages and one line of infor- mation per function that is compiled.</li> </ul>
		$\mathfrak{o}$ — Only warning messages are printed.
:verbose		-1 — Nothing at all is printed out.
		If this argument has any other value, the level 1 information is printed. The default value for :print is *compile-print*, which has the default value 1.
	:verbose	Controls the printing of messages about the current optimization settings. The default value is *compile-verbose*, which defaults to t.
Values	-	s returned, being the pathname argument . if the compile was unsuccessful
Description	compile-file calls the compiler to translate a Lisp source file into a form that both loads and runs faster. A compiled func-	

tion typically runs more than ten times faster than when interpreted (assuming that it is not spending most of its work calling already compiled functions). A source file with a .lisp extension compiles to produce a file with a .\*x\*fas1 extension (the actual extension depends on the host machine CPU). Subsequent use of load loads the compiled version (which is in LispWorks's FASL or Fast Load format) in preference to the .lisp source.

In compiling a file the compiler has to both compile each function and top level form in the file, and to produce the appropriate FASL directives so that loading has the desired effect. In particular objects need to have space allocated for them, and top level forms are called as they are loaded.

```
Examples (compile-file "devel/fred.lisp")

;; compile fred.lisp to fred.fasl
(compile-file "devel/fred")

;; does the same thing

(compile-file "test" :load t)

;; compile test.lisp, then load if successful

(compile-file "program" :output-file "program.abc")

;; compile "program.lisp" to "program.abc"

(push "abc" sys::*binary-file-types*)

;; tells LispWorks that files with extension

;; ".abc" are binaries

Notes See declare for a list of the declarations that alter the behav-
```

Notes See declare for a list of the declarations that alter the behavior of the compiler.

> The act of compiling a file should have no side effects, other than the creation of symbols and packages as the input file is read by the reader.

During compilation an output file with an extension of .fas1\_t is used, so that an unsuccessful compile does not overwrite an existing .fas1.

The actual extensions used vary according to machine type. See under with-output-to-fasl-file for a full list of FASL file extensions.

See also compile disassemble

# concatenate

Summary	Redefines the standard concatenate function allowing it to take any Common Lisp type.
Package	common-lisp
Signature	<pre>concatenate result-type &amp;rest sequences =&gt; result-sequence</pre>
Arguments	<i>result-type</i> A type specifier.
	sequences A sequence.
Values	result-sequence A sequence.
Description	The concatenate function has been extended to take any Common Lisp type. The <i>result-sequence</i> will be of type <i>result-</i> <i>type</i> unless this is not possible, in which case a type-error is signalled.
See also	coerce
declare	Special Form
Summary	Declares a variable as special, or provides advice to the Com- mon Lisp system.
Package	common-lisp

Signature	declare <i>declaration</i> *	
Arguments	<i>declaration</i> A declaration specifier, not evaluated.	
Values	The special form declare behaves computationally as if it not present (other than to affect the semantics), and is only allowed in certain contexts, such as after the variable list in let, do, defun, etc.	
	(Consult the syntax definition of each special form to see if it takes declare forms and/or documentation strings.)	
Description	There are two distinct uses of declare: one is to declare Lisp variables as "special" (this affects the semantics of the appropriate bindings of the variables), and the other is to provide advice to help the Common Lisp system (in reality the compiler) run your Lisp code faster, or with more sophisticated debugging options.	
Notes	Note the following LispWorks extensions to the Common Lisp definition of declare:	
	• lambda-list specifies the value to be returned when a programmer asks for the arglist of a function	
	• values specifies the value to be returned when you ask for a description of the results of a function	
	• invisible-frame specifies that calls to this function should not appear in a debugger backtrace	
	• alias specifies that calls to this function should be displayed as calls to an alternative function in a debugger backtrace	
	If you use declare to specify types (and so turn off type- checking for the specified symbols) and then supply the wrong type, you may obtain a "Segmentation Violation". You can check this by interpreting the code (rather than compil- ing it).	

See also	compile
	compile-file
	proclaim

# defpackage

# Macro

Summary	Remains as defined in Common Lisp, but see *handle- existing-defpackage* for an HCL extension.		
Package	common-lisp		
Signature	defpackage defined-package-name option &key add-use-defaults => package		
Arguments	defined-package-name		
		A string.	
	option	Keyword options.	
	add-use-defaults	A keyword	
Values	package	A package.	
Description	The macro defpackage is as defined in the ANSI standard, with the inclusion of the :add-use-defaults keyword. How- ever, the standard explicitly declines to define what defpack- age does if <i>defined-package-name</i> already exists and is in a state that differs from that described by the defpackage form.		
	Therefore an extension has been written that allows you to select between alternative behaviors. See <i>*handle-existing-defpackage*</i> for full details.		
	If <i>add-use-defaults</i> is specified with the argument <code>t</code> , the fol- lowing packages are included in the defined package:		
	• common-lisp		
	• lispworks		
	• harlequin-common-lisp		

See also \*handle-existing-defpackage\*

directory		Function	
Summary	Determines which files on the system have named matching a given pathname.		
Package	common-lisp	common-lisp	
Signature	directory path pathnames	nname &key test directories check-for-subs =>	
Arguments	pathname	A pathname, string, or file-stream.	
	test	Filtering test (only pathnames matching the test are collected).	
	directories	If non-nil then directories are included in the collection.	
	check-for-subs	If non-nil (the default), then pathnames representing directories have null name components.	
Values	pathnames	A list of physical pathnames.	
Description	directory coll pathname.	ects all the pathnames matching the given	
	Because truenames are now returned, the entries . and no longer show up in the output of directory. This means, for instance, that		
	(directory #P"/usr/users/")		
	does not include #p"/usr", which is the truename of #p"/usr/users/"		
	The specification is unclear as to the appropriate behavior of directory in the presence of links to non-existent files or		

directories. For example, if the directory contains foo, which is a symbolic link to bar, and there is no file named bar, should bar show up in the directory listing? A keyword argument has been added which lets you control this behavior.

Example CL-USER 15 > (pprint (directory "."))
 (#P"C:/Program Files/Harlequin/LispWorks/lib/"
 #P"C:/Program Files/Harlequin/LispWorks/lispworks 4100.exe"
 #P"C:/Program Files/Harlequin/LispWorks/readme.txt"
 #P"C:/Program Files/Harlequin/LispWorks/DeIsL1.isu"
 #P"C:/Program Files/Harlequin/LispWorks/MSVCRT.DLL")

# disassemble

Summary	Views the mach assembly langu	nine code of a compiled function definition in lage form.
Package	common-lisp	
Signature	disassemble na	nme-or-function => byte-size
Arguments	name-or-function	n Either a function-object, a lambda-expres- sion or a symbol with a function definition. If the function in question is not compiled (i.e. a lambda-expression) then it is first compiled using the function compile.
Values	byte-size	The size of the compiled code in bytes. An error is signalled if the argument to disas- semble is not suitable.
Description		iews the machine-code of a compiled function sembly language form.

Examples	(disassemble '(lambda (x) (progn x))) (disassemble 'cons) (disassemble #'map)
Notes	The output from disassemble lacks useful information such as local and lexical variable names and symbol names. The representation of integers or characters or Lisp objects in gen- eral is not easily readable without detailed knowledge of the internals of the Lisp system and the host machine instruction set.
See also	compile compile-file

# input-stream-p

Generic Function

Summary	A generic funct stream.	ion that determines if an object is an input
Package	common-lisp	
Signature	input-stream-p	stream => result
Arguments	stream	A stream.
Values	result	A generalized boolean.
Description	The predicate input-stream-p is implemented as a generic function. The default method returns t if <i>stream</i> is an input stream. If the user wants to implement a stream with no inherent directionality (and thus does not include fundamental-input-stream Or fundamental-output-stream) but for which the directionality depends on the instance, then a method should be provided for input-stream-p.	
See also	fundamental-in output-stream	-

# interactive-stream-p

Function

Summary	A generic funct tive stream.	ion that determines if an object is an interac-
Package	cl	
Signature	interactive-st	ream-p stream -> bool
Arguments	stream	A stream.
Values	bool	A generalized boolean.
Description	The predicate interactive-stream-p is implemented as a generic function. The fundamental-stream class provides a default method that returns nil.	
See also	input-stream-	-

# load-logical-pathname-translations

Summary	Searches for and loads the definition of a logical host, if not already defined.	
Package	cl	
Signature	<pre>load-logical-pathname-translations host =&gt; just-loaded</pre>	
Arguments	host	A logical host, expressed as a string.
Values	just-loaded	A generalized boolean
Description	This function loads the translations for <i>host</i> by loading the file <i>host</i> .lisp from the LispWorks directory translations.	

# Macro

# loop

Summary	A macro that performs iteration.		
Package	cl		
Signature	<pre>loop {for   as } var [type-spec]     being {the   each } {records   record }     {in   of } query-expression =&gt; result</pre>		
Arguments	var	A variable.	
	query-expression	An SQL query-statement	
Values	result	An object.	
Description	The Common Lisp loop macro has been extended with a clause for iterating over query results. This extension is available only when the SQL interface has been loaded. See Chapter 15, <i>Common SQL</i> . For a full description of the rest of the Common Lisp loop facility, see the Common Lisp Hyperspec.		
	Each iteration of the loop assigns the next record of the table to the variable <i>var</i> . The record is represented in Lisp as a list. Destructuring can be used in <i>var</i> to bind variables to specific attributes of the records resulting from <i>query-expression</i> . In conjunction with the panoply of existing clauses available from the loop macro, the new iteration clause provides an integrated report generation facility.		
Example	This extended loop example, on each record returned as a result of the query, binds name, finds the salary (if any) from an associated hash-table, and for salaries greater than 20000: increments a count, accumulates the salary, and prints the details. Finally, the average salary is printed.		

```
(loop
               for (name) being each record in
               [select [ename] :from [emp]]
               as salary = (gethash name *salary-table*)
               initially (format t "~&~20A~10D" 'name 'salary)
               when (and salary (> salary 20000))
                count salary into salaries
                and sum salary into total
                and do (format t "~&~20A~10D" name salary)
               else
                do (format t "~&~20A~10A" name "N/A")
               finally
                (format t "~2&Av Salary: ~10D" (/ total salaries)))
See also
             do-query
             map-query
             query
             select
```

# make-sequence

Summary	Redefines the standard make-sequence function allowing it to take any type specifier.	
Package	common-lisp	
Signature	make-sequence	result-type size &key initial-element => sequence
Arguments	result-type	A type specifier.
	size	A non-negative integer.
	initial-element	An object.
Values	sequence	A sequence.
Description	The make-sequence function has been extended to take any Common Lisp type. The <i>sequence</i> will be of type <i>result-type</i> unless this is not possible, in which case a type-error is signalled.	

See also	concatenate map merge	
map		Function
Summary	Redefines the s type specifier.	tandard map function allowing it to take any
Package	common-lisp	
Signature	map <i>result-type f</i>	function &rest sequences => result
Arguments	result-type	A sequence type specifier or nil.
	function	A function designator.
	sequence	A sequence.
Values	result	A sequence.
Description	Lisp type. The	on has been extended to take any Common <i>result</i> will be of type <i>result-type</i> unless this is which case a type-error is signalled.
See also	concatenate make-sequence merge	
merge		Function
Summary	Redefines the st type specifier.	tandard merge function allowing it to take any
Package	common-lisp	

Signature	merge <i>result-type</i> sequence	e sequence1 sequence2 predicate &key key =>
Arguments	result-type	A type specifier.
	sequence1	A sequence.
	sequence2	A sequence.
	predicate	A designator for a function.
	key	A designator for a function or nil.
Values	sequence	A sequence.
Description	The merge function has been extended to take any Common Lisp type. The <i>sequence</i> will be of type <i>result-type</i> unless this is not possible, in which case a type-error is signalled.	
See also	concatenate	
	make-sequence	
	map	
		<b>F</b> 4*
open		Function

Summary	Creates, oper specified file	ns, and returns a file stream that is connected to a
Package	common-lisp	
Signature	open filespec &key direction element-type external-format if-exists if- does-not-exist => stream	
Arguments	filespec	An external format spec, where the name can be :default.
	direction	If <i>direction</i> is :probe, <i>external-format</i> is ignored. The element type and external format of the returned stream are undefined.

	element-type	By default, the value of lw:*default- character-element-type* (the ANSI stan- dard default is character).	
	external-format	An external file format designator. By default, this is :default.	
	if-exists	What to do if the file stream already exists. The possible values for this are as in the ANSI standard.	
	if-does-not-exist	What to do if the file stream does not already exist. The possible values for this are as in the ANSI standard.	
Values	stream	A file stream, or nil.	
Description	If external-format has a name which is not :default and the parameters include :eol-style, it is used as is. Otherwise, the system decides which external format to use via guess-external-format. By default, this finds a match based on the filename; or (if that fails), looks in the EMACS- style (-*-) attribute line for an option called encoding or external-format; or (if that fails), chooses from among likely encodings by analyzing the bytes near the start of the file. By default, it then also analyses the start of the file for byte pat- terns indicating the end-of-line style, and uses a default end- of-line style if no such pattern is found. This behavior is con- figurable.		
After the external-format has been determined, it is ve using valid-external-format-p; an error is signalled check fails.		,	
	If open gets :default as its <i>element-type</i> arg, it chooses the type on the basis of the external format. If open gets an <i>element-type</i> other than :default and the direction is :input or :io, the argument must be a supertype of the type of characters produced by the external format; if the direction is :out-		

put or :io, it must be a subtype of the type of characters accepted by the external format; if it does not satisfy these requirements, an error is signalled.

See also	<pre>*default-character-element-type*</pre>
	guess-external-format
	valid-external-format-p

# open-stream-p

**Generic Function** 

Summary	A generic function that determines if a stream has been closed.
Package	common-lisp
Signature	open-stream-p stream => result
Arguments	stream A stream.
Values	<i>result</i> A generalized boolean.
Description	The function open-stream-p is generic. The default method provided by the class fundamental-stream returns t if close has not been called on the stream.
See also	close fundamental-stream

# output-stream-p

**Generic Function** 

Summary A generic function that determines if an object is an output stream.

Package common-lisp

Signature	output-stream-p stream => result	
Arguments	stream	A stream.
Values	result	A generalized boolean.
Description	function. The d stream. If the u inherent directi fundamental-i stream) but for	putput-stream-p is implemented as a generic efault method returns t if <i>stream</i> is an output ser wants to implement a stream with no ionality (and thus does not include nput-stream Or fundamental-output- which the directionality depends on the method should be provided for output-
See also	fundamental-o input-stream-	-
proclaim		Function
Summary	Established a sj ment.	pecified declaration in the global environ-

Cultural	ment.	
Package	common-lisp	
Signature	<pre>proclaim declaration-list =&gt; nil</pre>	
Arguments	<i>declaration-list</i> A list of declaration forms to be put into immediate and pervasive effect.	
Values	Returns nil.	
Description	Unlike declare, proclaim is a function that parses the declarations in the list (usually a quoted list), and puts their semantics and advice into global effect. This can be useful	

when compiling a file for speedy execution, since a proclamation such as:

(proclaim '(optimize (speed 3) (space 0) (debug 0)))

means the rest of the file is compiled with these optimization levels in effect. (The other way of doing this is to make appropriate declarations in every function in the file).

See the LispWorks User Guide for a more extended description.

Examples	(proclaim	'(special	<pre>*fred*))</pre>	
	(proclaim	'(type sin	gle-float x	y z))
	(proclaim	'(optimize	(safety 0)	(speed 3)))

Notes As proclaim involves parsing a list of lists of symbols and is intended to be used a few times per file, its implementation is not optimized for speed — it makes little sense to use it other than at top level.

> Remember to quote the argument list if it is a constant list. (proclaim (special x)) attempts to call function special.

Exercise caution if you declare or proclaim variables to be special without regard to the naming convention that surrounds their names with asterisks.

See also compile compile-file declare

### room

Summary	Print information about the state of internal storage and its management.
Package	common-lisp
Signature	room & optional x

Arguments	x One of mil, t, or :default.		
Values	room returns no values.		
Description	This function provides statistics on the current state of the storage, including the amount of space currently allocated, and the amount available for allocation.		
	As outlined in the Common Lisp Hyperspec, the room func- tion takes an optional argument which controls the level of detail it produces.		
	Given an argument of nil, a summary of the total allocation in the entire heap (in kilobytes) is produced. The "allocated" figure only represents the amount of space allocated in heap segments that are writable, as opposed to read-only segments that hold some of the system code such as the garbage collec- tor itself. The free space figure covers all the free space in all segments.		
	When called without an argument, <b>room</b> additionally prints information on the distribution of space between the genera- tions of the heap.		
	When called with argument <code>±</code> , a breakdown of allocation in the individual segments of each generation is produced. Each segment is identified by its start address in memory. For each segment there is a free space threshold (the "minimum free space")—when the available space in the segment falls below this value, the garbage collector takes action to attempt to free more space in this segment.		
	Two statistics about promotion are also reported on a per- segment basis: the number of sweeps that an object must sur- vive in this generation before becoming eligible for promo- tion, and the total volume of objects that have survived for that long and are consequently awaiting promotion to the next generation. These statistics are not relevant for static segments, which are indicated as "static".		

Examples USER 109 > (room nil) Total Size 10043K, Allocated 4898K, Free 2009K USER 110 > (room)Generation 0: Total Size 1314K, Allocated 35K, Free 12 71K Generation 1: Total Size 1284K, Allocated 731K, Free 5 08K Generation 2: Total Size 4376K, Allocated 4131K, Free 230K Total Size 10043K, Allocated 4898K, Free 2009K USER 111 > (room t)Generation 0: Total Size 1314K, Allocated 35K, Free 1 271K Segment 7203872: Total Size 256K, Allocated 35K, Free 216K minimum free space 64K, Awaiting promotion = 0K, sweeps before promotion =10 Segment 9200432: Total Size 1058K, Allocated OK, Free 1054K minimum free space OK, Awaiting promotion = 0K, sweeps before promotion =10 Generation 1: Total Size 1284K, Allocated 731K, Free 5 08K Segment 8151856: Total Size 1024K, Allocated 577K, Fre e 425K minimum free space OK, Awaiting promotion = 0K, sweeps before promotion =10 Segment 5631008: Total Size 132K, Allocated 109K, Free 1K minimum free space 3K, Awaiting promotion = 0K, sweeps before promotion =10 Segment 5499936: Total Size 128K, Allocated 43K, Free 80K minimum free space OK, static Generation 2: Total Size 4376K, Allocated 4131K, Free 230K Segment 7466016: Total Size 669K, Allocated 595K, Free 70K minimum free space OK, Awaiting promotion = 472K, sweeps before promotion =1 0 Segment 5767040: Total Size 1403K, Allocated 1316K, Fr ee 82K minimum free space OK, Awaiting promotion = 1174K, sweeps before promotion = 10

	Segment 3014656: Total Size 2304K, Allocated 2219K, Fr ee 78K
	minimum free space OK,
	Awaiting promotion = 2174K, sweeps before promotion =
	10
	Total Size 10043K, Allocated 4898K, Free 2009K USER 112 >
See also	find-object-size total-allocation

# simple-base-string

Type

Summary	The simple base string type.	
Package	common-lisp	
Signature	simple-base-string length	
Arguments	length	The length of the string (or <b>*</b> , meaning any).
Description	The type of simple base strings.	

# step

Macro Summary Steps through the evaluation of a form. Package common-lisp Signature step form => result Arguments A form to be stepped and evaluated. form Values The values returned by form. result Description step evaluates a form and allows you to single-step through it. You can include a call to step inside a tricky definition to

invoke the stepper every time the definition is used. step can also optionally step through macros.

The commands shown below are available. When certain stepper variables (as described below) are set, some of these commands are not relevant and are therefore not available. Use :help to get a list of the commands.

:s N	Step this form and all of its subforms (optional positive integer argument).
:st	Step this form without stepping its sub- forms.
:su	Step up out of this form without stepping its subforms.
:sr	Return a value to use for this form.
:sq	Quit from the current stepper level.
:redo	Redo one of the previous commands.
:get	Get an item from the history list and put it in a variable.
:help	List available commands.
:use	Replace one form with another form in pre- vious command and redo it.
:his	List the commands history.
The optional in	teger argument <i>n</i> for <b>:s</b> means do <b>:s</b> <i>n</i> times.

Examples The following examples illustrate some of these commands.

```
USER 12 > (step (+ 1 (* 2 3) 4))
(+ 1 (* 2 3) 4) -> :s
   1 -> :s
   1
   (* 2 3) -> :su
   6
   4 -> :s
   4
11
11
USER 13 > (defun foo (x y) (+ x y))
FOO
USER 14 > step (foo (+ 1 1) 2)
(FOO (+ 1 1) 2) -> :st
   (+ 1 1) -> :s
     1 -> :s
     1
      1 -> :s
     1
   2
   2 -> :s
   2
4
4
USER 15 > :redo (STEP (FOO # 2))
 (FOO (+ 1 1) 2) -> :s
    (+ 1 1) -> :s
       1 -> :s
       1
    2
    2 -> :s
    2
    (+ X Y) -> :s
       X -> :s
       2
       Y -> :s
       2
    4
 4
 4
```

```
Interaction
```

You can interact when an evaluated form returns, by setting the variable \*no-step-out\* to nil. The prompt changes as shown below:

```
USER 36 > \text{step} (\text{cons } 1 \ 2)
               (CONS 1 2) -> :s
                  1 -> :s
                  1 = 1 < -: sr 3
                  2 -> :s
                  2 = 2 < -: sr 4
               (CONS 1 2) = (3 . 4) < -: s
               (3.4)
Macro
               To allow expansion of macros, set the variable *step-mac-
               ros* t0 t.
expansion
Stepping
               To step through the function calls in compiled code, set the
compiled
               variable hcl:*step-compiled* to t.
code
Showing the
               If required, the stepper can print out the step level: set the
stepper level
               variable *print-step-level* to t.
               USER 21 > (setg *print-step-level* t)
               т
               USER 22 > \text{step} (\text{cons } 1 \ 2)
               [1](cons 1 2) \rightarrow :s
                     1 -> :s
               [2]
                                     1
               [2] 2 -> :s
                     2
                  (1.2)
               (1.2)
               It is not advisable to try and step certain compiled functions,
Non-steppa-
ble functions
               such as car and format. The variable hcl:*step-filter*
               contains a list of functions which should not be stepped. If
               you get deep stack overflows inside the stepper, you may
               need to add a function name to hcl:*step-filter*.
Printing
               By default, the stepper uses the same printing environment
               as the rest of LispWorks (the same settings of the *print-
               ...* variables). To control the stepper printing environment
               independently, set the variable hcl:*step-print-env* to t.
               The values of the variables hcl:*step-print-...* are then
               used instead of the variables *print-...*.
```

# stream-element-type

# Generic Function

Summary	Implements stream-element-type as a generic function.		
Package	common-lisp		
Signature	stream-element	-type stream => type	
Arguments	stream	A stream.	
Values	type	A type specifier.	
Description	The function stream-element-type is implemented as a generic function. Depending on the stream, a method should be defined for this generic function that returns the element type of the stream.		
	method is prov	ndamental-character-stream a default ided which returns CHARACTER. A method led for stream classes based on the fundamen- ream class.	
See also	fundamental-b fundamental-c	inary-stream haracter-stream	
string		Туре	
Summary	The string type		
Package	common-lisp		
Signature	string length ele	ement-type	
Arguments	length	The length of the string (or <b>*</b> , meaning any).	

	Va	he type of string element. The defau alue of lw:*default-character-ele ype* rather than *.		
Description	The union of all string types as specified in the standard extended with an extra parameter: <i>element-type</i> .			
	(string length element-type) means all string types whose element type is a subtype of element-type. That is:			
	(string * base-ch	har) = (vector base-char *)		
	(string * lw:sim	ple-char) = (or (vector base-char (vector lw:simple-cl	-	
	(string * charact	ter) = (or (vector base-chan (vector lw:simple-o (vector character	char *)	
Example	CL-USER 235 > (lw:set-default-character-element-type 'base-char)			
	BASE-CHAR CL-USER 236 > (co "foo"	oncatenate 'string "f" "o" "o")		
	CL-USER 237 > (ty SIMPLE-BASE-STRIM			
See also	*default-character-element-type* set-default-character-element-type			

# time

# Macro

Summary	Determines the ronment.	execution time of a form in the current envi-
Package	common-lisp	
Signature	<pre>time form =&gt; result</pre>	
Arguments	form	A form to be evaluated

Values	result Th	e result of the evaluation of the form.
Description	evaluates the form some timing and si onds), the elapsed	o determine execution times. The macro given to it as argument, and prints out ze data: the user and system times (in sec- time (in hours, minutes and seconds), and heap space allocated in executing the
	0	e data covers all stack groups, not just the me. Note that time itself uses a small, con- ap space.
Examples	USER 10 > (time ( startend user time = system time = Elapsed time = 0 Allocation = 7880 NIL	
See also	extended-time	
trace		Macro
Summary	Invoke the Commo tions.	on Lisp tracing facility on the named func-
Package	common-lisp	
Signature	trace {function-nam	e tracing-desc}* => trace-result
	tracing-desc ::= (ds	pec {keyword form}*)
	dspec ::= function (meth	on-name   od generic-function-name [qualifier] (class*))

	:e :b :ir :w	ter  :allocation   :before   :backtrace   val-after   :eval-before   :break   reak-on-exit   :entrycond   :exitcond   uside   :process   :trace-output   :step   hen
	quaimer ::= :at	ter   :before   :around
Arguments	function-name	A symbol whose symbol-function is to be traced. Functions, macros and generic func- tions may be specified this way.
	dspec	Specifies the functional definition which is to be traced. This either has the same form as above, or specifies a method by the name of its generic function and by a list of classes to specialize the arguments to the method. In this latter case the list of classes must cor- respond exactly to the classes of the special- ized parameters of an existing method, and then only this method is traced (as opposed to the corresponding generic function).
	tracing-desc	Specifies the functional definition which is to be traced and specifies any additional options that are required.
	upon returning are also printed	yed by a list of forms; these are evaluated from the function. The values of these forms out by the tracer. The forms are evaluated at the moults of the function call, and if they

upon returning from the function. The values of these forms are also printed out by the tracer. The forms are evaluated after printing out the results of the function call, and if they modify hcl:\*traced-results\* then the values received by the caller of the function are correspondingly altered (see also hcl:\*traced-results\*).

:allocation — if non-nil, the memory allocation made during a function-call is printed upon exit from the function. This allocation is counted in bytes. If it is any other symbol (except nil), trace uses the symbol to accumulate the amount of allocation made between entering and exiting the function. Upon exit from the function, the symbol contains the number of bytes allocated during the function-call. For example,

```
(trace (print :entrycond nil
                :exitcond nil
               :allocation $$print-allocation))
```

results in \$\$print-allocation containing the sum of the
allocation made inside print.

Note that if the function is called again, trace continues to use \$\$print-allocation as an accumulator of memory allocation. It adds to the present value rather than re-initializing it each time the function is called.

:backtrace generates a backtrace on each call to the traced function. It is followed by a keyword that can be any of the following values:

:quick	Like the :pd debugger command.
t	Like the :ь debugger command.
:verbose	Like the :b :verbose debugger command.
:bug-form	Like the :bug-form debugger command.

:before is followed by a list of forms; these are evaluated upon entering the function and their values are printed out by the tracer. The forms are evaluated after printing out the arguments to the function, and if they alter hcl:\*tracedarglist\* then the values received by the body of the function are changed accordingly (see also \*traced-arglist\*).

:eval-after and :eval-before are similar to :after and :before, without output.

:break is followed by a form. This is evaluated after printing the standard information caused by entering the function, and after executing any :before forms; if it returns nil then tracing continues normally, otherwise break is called. This provides a way of entering the debugger through the tracer. :break-on-exit is followed by a form. This is evaluated after printing the standard information caused by returning from the function, and before executing any :after forms; if it returns nil then tracing continues normally, otherwise break is called. This provides a second way of entering the debugger through the tracer.

:entrycond controls the printing of the standard entry message (including the function's arguments). If the form following it evaluates to give a non-nil value when the function is entered, then the entry message is printed (but otherwise it is not). If this option is not present then the standard entry message is always printed upon calling the function. See also the :when option.

:exitcond controls the printing of the standard exit message (including the function's results). If the form following it evaluates to give a non-nil value when the function is exited, then the exit message is printed (but otherwise it is not). If this option is not present then the standard exit message is always printed upon returning from the function. See also the :when option.

:inside restricts the tracing to within one of the functions given as an argument. A single symbolic function name is treated as a list of one element, i.e. :inside format is equivalent to :inside (format).

:process may be used to restrict the tracing to a particular process. If it is followed by a process then the function is only traced when it is invoked from within that process. If it is followed by t then it is traced from all processes — this is the default. In any other cases the function is not traced at all.

:trace-output should be followed by a stream. All the output from tracing the function is sent to this stream. By default output from the tracer is sent to \*trace-output\*. Use of this argument allows you to dispatch traced output from different functions to different places.

:step, when non-mil, invokes the stepper (for evaluated functions).

:when overrides all other keywords. It is followed by an expression, and tracing only occurs when that expression evaluates to non-nil. It is useful if you want to combine :entrycond and :exitcond.

Values trace-result If trace is called with no arguments then it returns a list of the names of all the functions currently being traced. When called with one or more arguments, it returns the symbols of the functions specified in those arguments.

Description trace is the macro used to invoke the tracing facility. This is a useful debugging tool that enables information about selected calls to be generated by the system. The standard way of invoking trace is to call it with the names of the functions, macros and methods that are to be monitored in this way. Calls to these produce a record of the function that was called, the arguments it received and the results it produced.

> The arguments to trace each specify a function (or a macro or a method) to be traced. They may also contain further instructions to control how the tracing output is displayed, or to cause particular actions to occur when the functions is called or exited. If trace is called with a function that is already being traced, then the new tracing specification for that function replaces the old version.

```
Example 1 USER 1 > (defvar *number-of-calls-to-max* 0)
*NUMBER-OF-CALLS-TO-MAX*
USER 2 > (trace (max :after
((incf *number-of-calls-to-max*))))
(MAX)
```

```
USER 3 > (dotimes (i 2) (max i 1))
             0 MAX > (0 1)
             0 MAX < (1)
              1
             0 MAX > (1 1)
             0 MAX < (1)
              2
             NIL
             USER 4 > *number-of-calls-to-max*
             2
             USER 5 > (trace (max
                               :entrycond
                               (> (length compiler:*traced-arglist*)
                                  2)
                               :exitcond nil))
             (MAX)
             USER 6 > (max 2 3 (max 4 5))
             0 MAX > (2 3 5)
             5
Example 2
             This example illustrates the use of :inside.
             CL-USER 2 > (defun outer ()
                           (inner))
             OUTER
             CL-USER 3 > (defun inner ()
                           10)
             INNER
             CL-USER 4 > (trace (inner :inside outer))
                                 ;; only trace when inside OUTER
             (INNER)
             CL-USER 5 > (inner)
                 ;; no tracing occurs since we are not inside OUTER
             10
             CL-USER 6 > (outer) ;; INNER is traced inside OUTER
             0 INNER > NIL
             0 \text{ INNER} < (10)
             10
             CL-USER 7 >
```

Example 3 To trace a method:

	(defmethod foo (x) x) (trace ((method foo (t))))
See also	*trace-output* untrace

# \*trace-output\*

Summary	The stream on v print their outp	vhich traced functions and the time macro ut.
Package	common-lisp	
Initial Value	A synonym stre *terminal-io*.	am that uses the stream that is the value of
Description	The value of *trace-output* is used as the stream to which tracing sends its output by default. Other streams can be explicitly specified by the :trace-output option to trace.	
See also	trace	
untrace		Macro
untrace Summary	Turns off the Co functions.	<i>Macro</i> ommon Lisp tracing facility on the named
Summary	functions. common-lisp	
Summary Package	functions. common-lisp	ommon Lisp tracing facility on the named

Values	When called with no arguments, it returns the symbols of all functions currently being traced. When called with one or more functions as arguments, untrace returns a list of the symbols of those functions. Thus, in all situations, untrace returns a list of the symbols of those functions being untraced.
Description	untrace is used to cease the tracing of functions. If it is called with no arguments then the tracing of all currently traced functions is stopped. If it is called with one or more symbols then the tracing of those functions is stopped. A warning is given if untrace is called with a function that is not being traced.
Examples	USER 12 > (progn (untrace) (trace + - / *)) *
	USER 13 > (+ 2 3) 0 + > (2 3) 0 + < (5) 5
	USER 14 > (untrace + -) (* $ / $ )
	USER 15 > (+ 2 3) 5
	To untrace a method:
	(untrace (clos:method foo (t)))
See also	trace

# The COMPILER Package

This chapter describes symbols available in the COMPILER package.

#### deftransform

Macro

Summary	Defines a function that computes the expansion of a form.	
Package	compiler	
Signature	deftransform name transform-name lambda-list &body body => list-of-transforms	
Arguments	name	A symbol naming the function to which the transform is to be applied.
	transform-name	The symbol naming the transformation — it should be unique for the function being transformed — and provides a handle with which to redefine an existing transform.
	lambda-list	This must match against the form being expanded before expansion is allowed to take place, in the sense that it must be valid

		to call a function with such a lambda-list using the arguments supplied in the candi- date-form for expansion.
	body	The body of the expander function, the result of which replaces the original form (unless it evaluates to compiler::%pass%, in which case no transformation is applied).
Values	list-of-transforms	A list of the names of transforms defined for the function, including the one just added.
Description	putes the expan the compiler an	ike defmacro, defines a function that com- sion of a form. Transforms are only used by d not by the interpreter. deftransform allows a optimizations performed by the compiler.
Examples	`(system:: +2 (compiler:deft	ransform + +-of-2 (x y)   ,x ,y)) ransform + +-of-many (x &rest y)   ,x (+ ,@y)))
	<pre>;; compiles t ;; of + (inlight)</pre>	ression like (+ a b c 4 5 7 d e f) o use the binary version ned by default), n the full (slow) version of +
	(compiler:deft `(cons ,x '()	
	(compiler:deft `(cons ,x (co	ransform list list-of-2 (x y) ns ,y '())))
		g to call list - lined by default
	(cond ((	<pre>ransform constant my-trans (x) constantp x) x) consp x) `(quote ,(eval x))) 'compiler::%pass%))) ; give up if not a</pre>
	(compile (defu	n three-list () (constant (list 1 2 3))))

	<pre>;; the function three-list returns the ;; same list (1 2 3) ;; every time it is called</pre>
	The list-of-2 example returns
	(LIST-OF-2 LIST-OF-1 COMPILER::LIST-TRANSFORM)
	as its result, since a similar transform already exists in the compiler. by the name compiler::list*-transform.
Notes	deftransform differs from defmacro in various ways:
	The evaluation of the body can return compiler:%pass% indi- cating that the form is not to be expanded (in other words, the transform method has elected to give up trying to improve the code).
	The compiler only calls the expander function if the arguments match the lambda list — macros are unconditionally expanded.
	There can be several deftransforms for the same symbol, each having a different name. (The compiler calls each one in turn until one succeeds. This repeats until they all pass, so that the replacement form may itself be transformed.)
	If a transform takes keyword arguments the compiler pre- serves the correct order of evaluation.
	A carelessly written deftransform may lead the compiler to transform valid Common Lisp into incorrect code — there is no semantic checking of the transform.
See also	compile compile compile-file compile-file

The COMPILER Package

# The DBG Package

This chapter describes symbols available in the DBG package, used to configure the debugging information produced by LispWorks.

#### \*debug-print-length\*

Summary	Controls the number of object components printed in debug- ger output.
Package	dbg
Initial Value	40
Description	This variable is used to control the number of components of an object which are printed during output from the debug- ger. If its value is a positive integer then the components up to that number are printed. If it is nil then all the parts of an object are shown.
Examples	USER 83 > (setq dbg:*debug-print-length* 3)

#### \*debug-print-level\*

Notes

Summary	Controls the depth to which nested objected are printed in debugger output.
Package	dbg
Initial Value	4
Description	dbg:*debug-print-level* controls the depth to which nested objects are printed during output from the debugger. If its value is a positive integer then components at or above that level are printed. By definition an object to be printed is considered to be at level 0, its components are at level 1, their subcomponents are at level 2, and so on. If dbg:*debug- print-level* is nil then objects are printed to arbitrary depth.
Example	USER 89 > (setq dbg:*debug-print-level* 2)

	2 USER 90 > (subseq 3 '(cat (dog) ((goldfish)) (((hamster)))))
	Error: Illegal START argument (CAT (DOG) ((GOLDFISH)) (((HAMSTER)))) 1 (abort) return to top loop level 0.
	Type :c followed by a number to proceed USER 91 : $1 > :v$
	Call to CHECK-START-AND-END : Arg 0 (START): (CAT (DOG) (#) (#)) Arg 1 (END): NIL
Notes	*debug-print-level* is an extension to Common Lisp.

# \*hidden-packages\*

Summary	A list of packages whose symbols should not be displayed in debugger output.
Package	dbg
Initial Value	A list containing the dbg and conditions packages.
Description	dbg:*hidden-packages* is used by the debugger. It should be bound to a list of packages. If a package is included in the list then any symbols in it are not shown by the debugger. Thus during backtraces the call frames corresponding to functions in these packages are not displayed. This can be useful in restricting the debugger to particular areas.

Examples USER 108 > (setq dbg:\*hidden-packages\* (cons (find-package 'Lisp) dgb:\*hidden-packages\*)) (#<The LISP package, 10/224 internal, 829/911 external> #<The DBG package, 865/905 internal,</pre> 0/11 external> #<The CONDITIONS package, 577/704 internal,</pre> 89/111 external>) USER 109 > (cons unbound-var (u v)) Error: the variable UNBOUND-VAR is unbound. 1 (continue) Try evaluating it again 2 Return a value to 3 Return a value to set it to 4 (abort) return use to top loop level 0. Type :c followed by a number to proceed USER 110 : 1 > :b 3 Catch frame: (NIL) Catch frame: #: block-catcher-1748 Call to %EVAL : Call to %EVAL : USER 111 : 1 > Notes \*hidden-packages\* is an extension to Common Lisp.

#### \*print-binding-frames\*

Summary	Controls whether binding frames are printed in debugger output.
Package	dbg
Initial Value	nil
Description	This variable is used by the debugger when it displays the stack frames. Binding frames are formed when special vari- ables are bound, but are normally not shown by the debug-

	ger. However if $dbg:*print-binding-frames*$ is set to t then the binding frames are shown.
Example	<pre>USER 10 &gt; (defun print-to-length (object length)   (let ((*print-length* length))   (prinnt object)))</pre>
	PRINT-TO-LENGTH USER 11 > (setq dbg:*print-binding-frames* t)
	T USER 12 > (print-to-length '(1 2 3 4 5) 2)
	<pre>Error: Undefined function: PRINNT , with args ((1 2))</pre>
	1 (continue) Call PRINNT again 2 (abort) return to top loop level 0.
	Type :c followed by a number to proceed
	USER 13 : 1 > :n print-to-length
	USER 14 : 1 > :b :verbose 2 Interpreted call to (DEFUN PRINT-TO-LENGTH): *PRINT-LENGTH* : %INTERNAL-SPECIAL-MARKER% LENGTH : 2 OBJECT : (1 2 3 4 5) Binding frame: %BENV%,NIL
Notes	*print-binding-frames* is an extension to Common Lisp.

## \*print-catch-frames\*

Summary	Controls whether catch frames are printed in debugger output.
Package	dbg
Initial Value	t
Description	This variable is used by the debugger when it displays the stack frames. Catch frames are created when the special form catch is used. They are set up so that throws to the matching

	tag can be received. By default the debugger displays these frames, but if *print-catch-frames* is set to nil then the catch frames are no longer shown.
Examples	USER 17 > (setq dbg:*print-catch-frames* nil)
	<pre>NIL USER 18 &gt; (defun catch-it () (catch 'tag (throw-it) (print "Not caught"))) CATCH-IT USER 19 &gt; (defun throw-it () (throw 'tag (break)))</pre>
	THROW-IT USER 20 > (catch-it)
	break 1 (continue) return from break. 2 (abort) return to top loop level 0.
	Type :c followed by a number to proceed
	USER 21 : 1 > :b 5 Interpreted call to (DEFUN THROW-IT): Call to *%APPLY-INTERPRETED-FUNCTION : Interpreted call to (DEFUN CATCH-IT): Call to *%APPLY-INTERPRETED-FUNCTION : Call to %EVAL :
Notes	*print-catch-frames* is an extension to Common Lisp.

## \*print-handler-frames\*

Summary	Controls whether handler frames are printed in debugger output.	
Package	dbg	
Initial Value	nil	
Description	This variable is used by the debugger when it displays the stack frames. Handler frames are created by error handlers	

```
(see the LispWorks User Guide), and are normally not shown
             by the debugger. However if *print-handler-frames* is set
             to t then the handler frames are displayed.
Example
             USER 162 > (setq lw:*print-handler-frames* t)
             т
             USER 163 > (defun test (n)
                 (handler-case (fn-to-use n)
             (type-error () (format t "~%Type error~%") 0)))
             TEST
             USER 164 > (test #C(1 1))
             Error: Undefined function: FN-TO-USE, with args
               (\#C(1 \ 1))
             1 (continue) Call FN-TO-USE again
               2 (abort) return to top loop level 0.
             Type :c followed by a number to proceed
             USER 165 : 1 > :b 10
             Catch frame: (NIL)
             Catch frame: #: block-catcher-1854
             Call to *%UNDEFINED-FUNCTION-FUNCTION :
             Call to %EVAL :
             Call to RETURN-FROM :
             Call to %EVAL :
             Call to EVAL-AS-PROGN :
             Handler frame: ((TYPE-ERROR %LEXICAL-CLOSURE%
              (LAMBDA
               (CONDITIONS::TEMP)
               (GO #: |lambda-633|))
              ((#:|lambda-632|) (N . #))
              NIL ((#:|lambda-631|) (TEST))
              ((#:|lambda-633| # #))))
             Catch frame: "<* Catch All Object *>"
             Call to LET :
Notes
             *print-handler-frames* is an extension to Common Lisp.
```

### \*print-open-frames\*

Variable

Summary	Controls whether open frames are printed in debugger output.		
Package	dbg		
Initial Value	nil		
Description	This variable is used by the debugger when it displays the stack frames. Open frames are made by the system and are normally not shown by the debugger. However if *print-open-frames* is set to t then the open frames are displayed. It is unlikely that you need to examine open frames: their use is connected with implementation details.		
Examples	<pre>USER 52 &gt; (setq dbg:*print-open-frames* t) T USER 53 &gt; (car 2) Error: Cannot take CAR of 2 1 (abort) return to top loop level 0. Type :c followed by a number to proceed USER 54 : 1 &gt; :b 3 Open frame (5) Open frame (5) Call to CAR-FRAME :</pre>		
Notes	*print-open-frames* is an extension to Common Lisp.		

# \*print-restart-frames\*

Variable

Summary	Controls whether restart frames are printed in debugger out-
	put.

Package dbg

Initial Value nil

Description This variable is used by the debugger when it displays the stack frames. Restart frames are formed when restarts are established (see the *LispWorks User Guide*), but are normally not shown by the debugger. However if \*print-restart-frames\* is set to t then the restart frames are shown.

Example USER 43 > (setq dbg:\*print-restart-frames\* t) т USER 44 > (truncate 12.5 0.0)Error: Division-by-zero caused by TRUNCATE of (12.5 0.0) 1 (continue) Return a value to use 2 Supply new arguments to use 3 (abort) return to top loop level 0. Type :c followed by a number to proceed USER 45 : 1 > :b 5 Restart frame: (ABORT) Catch frame: (NIL) Catch frame: #: block-catcher-3223 Call to DIVISION-BY-ZERO-ERROR : Call to TRUNCATEANY : USER 46 : 1 >Notes \*print-restart-frames\* is an extension to Common Lisp.

85

The DBG Package

# The DSPEC Package

This chapter describes symbols available in the DSPEC package.

Definition specs, or dspecs, are a systematic way of naming definitions, that are intended to cover all kinds of definitions. One important use is keeping track of the locations of definitions. Simple examples of dspecs are:

```
car
(setf car)
(defclass standard-object).
```

#### \*active-finders\*

Summary	Controls how the source finding commands Find Source and Find Tag operate.	
Package	dspec	
Syntax	*active-finders*	
Description	The <b>*active-finders</b> * variable controls how the source finding commands, Find Source and Find Tag, operate. The legal values for the elements of this list are:	

:internal	The internal database of definitions per- formed in this image.	
:tags	Prompt for a tags file, when first used.	
pathname	Either a tags file or a tags database.	

A tags database is a file generated by save-tags-database.

The order of this list determines the order that the results from the finders are combined in — you would usually want <code>:internal</code> to be the first item on this list, as it contains the up-to-date information about the state of the image. More than one pathname is allowed.

#### save-tags-database

Summary	Saves the current internal dspec database to a given file.		
Package	dspec		
Syntax	save-tags-database pathname =>		
Arguments	pathname A filename		
Values			
Description	The save-tags-database function saves the current internal dspec database into the file given by <i>pathname</i> . The file can then be used in the variable *active-finders*.		
See also	*active-finders*		

# The EXTERNAL-FORMAT Package

This chapter describes symbols available in the EXTERNAL-FORMAT package.

#### char-external-code

Summary	Returns the code of a character in the specified character set.		
Package	external-format		
Signature	char-external-code charset => code		
Arguments	char	The character whose code you wish to return.	
	set	A character set. Legal values for <i>set</i> are :unicode, :latin-1, :ascii, :jis-x-208, :jis-x-212, :euc-jp, and :sjis.	
Values	code	The code of <i>char</i> in the character set <i>set</i> . An integer.	

Description Returns the code of the character *char* in the coded character set specified by *set*, or nil, if there is no encoding. Note that a coded character set is not the same thing as an external format.

For the *set* parameter, the :jis-\* codes are KUTEN indexes (from the 1990 version of these standards) encoded as

```
(+ (* 100 row) column)
```

:euc-jp is the complete two-byte format encoded as

```
(+ (* 256 first-byte) second-byte)
```

:sjis is Shift-JIS encoded in the same way. Strictly speaking, EUC and Shift-JIS are not coded character sets, but encodings of the JIS sets, but the encoding is easily expressed as an integer, so the same interface to it is used.

#### external-format-error

#### Condition

Summary	The condition class external-format-error is the superclass of all errors relating to external formats.
Package	external-format
Superclasses	error
Slots	<i>name</i> The name of the external format involved.
Description	The class external-format-error provides a slot for the name of external format involved: this is the fully expanded form of the specification with all the parameters filled in. It is also useful for users who want to set up a handler for encoding errors.

#### external-format-foreign-type

Summary

Package

Signature

Returns a type specifier for the integers handled by a speci- fied external format.
external-format
external-format-foreign-type external-format => type-specifier

- external-format An external character format. Arguments
- Values type-specifier A type specifier describing the integer types handled by *external-format*.
- Description Takes the name of an external format, and returns a Lisp type specifier for the type of integers that the external format handles on the foreign side.
- See also external-format-type

#### external-format-type

#### Function

Summary	Returns a type s fied external for	specifier for the characters handled by a speci- mat.
Package	external-forma	t
Signature	external-forma	t-type external-format => type-specifier
Arguments	external-format	An external character format.
Values	type-specifier	A type specifier describing the character types handled by <i>external-format</i> .

#### The EXTERNAL-FORMAT Package

Description	Takes the name of an external format, and returns a type		
	specifier for the type of characters that the external format		
	handles on the Lisp side.		

See also external-format-foreign-type

#### find-external-char

Summary	Returns the charset.	racter of a given code in a specified character
Package	external-format	
Signature	<pre>find-external-char code set =&gt; char</pre>	
Arguments	code	A character code. This is an integer.
	set	A character set. Legal values for <i>set</i> are :unicode, :latin-1, :ascii, :jis-x-208, :jis-x-212, :euc-jp, and :sjis.
Values	char	The character represented by <i>code</i> . If <i>code</i> is not a legal code in the specified set, the return value is undefined.
Description	Returns the character that has the code <i>code</i> (an integer) in the coded character set specified by <i>set</i> , or nil, if that character is not represented in the Lisp character set. Note that a coded character set is not the same thing as an external format. For the <i>set</i> parameter, the :jis-* codes are KUTEN indexes (from the 1990 version of these standards) encoded as	
	(+ (* 100 row)	column)
	:euc-jp is the c	omplete two-byte format encoded as
	(+ (* 256 firs	t-byte) second-byte)

:sjis is Shift-JIS encoded in the same way. Strictly speaking, EUC and Shift-JIS are not coded character sets, but encodings of the JIS sets, but the encoding is easily expressed as an integer, so the same interface to it is used.

See also char-external-code

#### valid-external-format-p

Summary	Tests whether an external format spec is valid.	
Package	external-format	
Signature	valid-external-format-p ef-spec &optional env => bool	
Arguments	ef-spec	An external format spec.
	env	An environment across which the spec should apply.
Values	bool	t if <i>ef-spec</i> is a valid spec; nil otherwise.
Description	This predicate tests whether the external format spec given in <i>ef-spec</i> is valid (in the environment <i>env</i> ).	
Example	<pre>(valid-external-format-p '(:Unicode :eol-style :lf))</pre>	

The EXTERNAL-FORMAT Package

# The HCL Package

This chapter describes symbols available in the HCL package, which is used by default when LispWorks is first loaded.

#### add-special-free-action

Summary	Adds a function to perform a special action during garbage collection.	
Package	hcl	
Signature	add-special-f	<pre>Eree-action function =&gt; function-list</pre>
Arguments	function	The function to be performed.
Values	function-list	A list of the functions currently called to perform special actions, including the one just added.
Description	"special action	bjects are garbage collected, you may require a n" to be performed as well. add-special- adds a function to perform the special action.

Note that the function is applied to all objects flagged for spe-
cial-free-action, so the function should check for the object's
type, so that it only affects relevant objects.
The functions flag-special-free-action and flag-not-
special-free-action flag and unflag objects for action.

Example (add-special-free-action 'free-my-app) See also remove-special-free-action flag-special-free-action flag-not-special-free-action

#### add-symbol-profiler

#### Function

Summary	Adds a symbol to the list of profiled symbols.	
Package	hcl	
Signature	add-symbol-pro	filer symbol => nil
Arguments	symbol	A symbol to be added to the *profile-symbol-list*.
Values	Returns nil.	
Description		ofiler adds a symbol to *profile-symbol- f profiled symbols.
See also	*profile-symbol- remove-symbol-	

#### allocation-in-gen-num

Macro

Summary Allocates objects from a specified generation within the scope of evaluating a number of forms.

Package	hcl		
Signature	allocation-in-gen-num gen-num &body body => result		
Arguments	gen-num	An integer, which if out of range for a valid generation number is rounded either to the youngest or oldest generation.	
		If <i>gen-num</i> is negative, the specified genera- tion is: the highest generation number + 1 – <i>gen-num</i> , so that an argument of –1 specifies the highest generation number.	
	body	The forms to be evaluated while the alloca- tion generation has been temporarily set to <i>gen-num</i> .	
Values	result	The result of evaluating body.	
Description	Allocates objects from a specified generation during the extent of the evaluation of the body forms.		
Examples	(allocation-in-gen-num 1		
	(setq tab (make-hash-table :size 1200 :test 'eq) arr (make-array 20)))		
Notes	Normally objects are allocated from the first (youngest) gen- eration, which assumes that they are short-lived. The storage allocator and garbage collector perform better if allocation of large numbers of non-ephemeral objects is done explicitly into a generation other than the youngest.		
See also	set-default-generation get-default-generation *symbol-alloc-gen-num*		

avoid-gc	Function	
Summary	Avoids garbage collection if possible.	
Package	hcl	
Signature	avoid-gc => previous-results	
Arguments	None.	
Values	The function returns the previous settings of minimum-for- sweep, maximum-overflow and minimum-overflow. (See set- gc-parameters for details of these.)	
Description	avoid-gc sets various internal parameters so that garbage collection is avoided as far as possible.	
	This can be useful with non-interactive programs.	
	If you use avoid-gc, use normal-gc later to reset the parame- ters to their default settings.	
See also	gc-if-needed normal-gc set-gc-parameters without-interrupts	
calls-who	Function	
Summary	Lists functions called by a function.	
Package	hcl	
Signature	calls-who <i>function</i> => callees	
Arguments	<i>function</i> A function whose callees are required.	
Values	<i>callees</i> A list of function called by function.	

Description	calls-who returns a list of the functions called by function
	(See also the editor commands List Callees, and show
	Paths From.)

See also who-calls

## change-directory

Summary	Changes the current directory.
Package	hcl
Signature	change-directory directory => current-dir
Arguments	<i>directory</i> A symbol or string specifying the new directory.
Values	change-directory returns the current directory.
Description	change-directory changes the current directory. Use *cur- rent-working-directory* to find the current directory.
clean-down	Function
Summary	Frees memory and reduces the size of the image, if possible.
Package	hcl
Signature	clean-down &optional free-upper => new-size
Arguments	The keyword :free-upper controls whether the empty seg- ments of the highest generation are removed or not. The default is t.
	ueraun is t.

Description	Tries to free as much memory as possible and then reduce the size of the image as much as possible, and also move all the allocated objects to generation 3. This is called by default before saving an image. It calls mark-and-sweep on generation 3, promotes all the objects into generation 3, deletes the empty segments and tries to reduce the image size. It may fail to delete empty segments if there are static segments in
	high address space.

See also save-image

## clean-generation-0

Summary	Attempts to promote all objects from generation zero into generation one, thereby clearing generation zero.
Package	hcl
Signature	clean-generation-0 => 1
Arguments	None
Values	Returns the value 1.
Description	This is useful when passing from a phase of creating long- lived data to a phase of mostly ephemeral data, for example, the end of loading an application and the start of its use.
Example	<pre>; allocate lots of non-ephemeral objects ; (clean-generation-0)</pre>
Notes	It may be more efficient to directly allocate the objects in a particular generation in the first place, using allocation-in- gen-num Or set-default-generation.

See also	allocation-in-gen-num
	collect-generation-2
	collect-highest-generation
	expand-generation-1
	set-promotion-count

## collect-generation-2

### Function

Summary	Controls wheth	er generation 2 is garbage collected.
Package	hcl	
Signature	collect-genera	ation-2 on => size
Arguments	on	If <i>on</i> is nil, generation 2 is not garbage collected. If <i>on</i> is t, the generation is garbage collected.
Values	size	The current size of the image.
Description		er generation 2 is garbage collected. (Genera- holds long-lived objects created dynami-
See also	clean-generat collect-highe expand-genera set-promotion	st-generation tion-1

## collect-highest-generation

### Function

Summary Controls whether the top generation is garbage-collected.

Package hcl

Signature collect-highest-generation flag

The HCL Package

Arguments	flag	If <i>flag</i> is non-nil, the top generation is collected; if <i>flag</i> is any other value, the top generation is not collected. The default is nil.
Values	collect-highe	st-generation returns no values.
See also	avoid-gc clean-generat collect-gener expand-genera normal-gc	ation-2
current-stack	-length	Function
Summary	Returns the size	e of the current stack.
Package	hcl	
Signature	current-stack-	<pre>length =&gt; stack-size</pre>
Arguments	None	
Values	stack-size	The current size of the stack, in words.
Example	(current-stack	c-length) => 16000

## \*default-package-use-list\*

Variable

Summary List of packages that newly created packages use by default.

Package hel

Initial Value ("CL" "LW" "HCL")

Description This variable is the default value of the :use keyword to defpackage, which specifies which existing packages the package being defined inherits from.

### \*default-profiler-limit\*

Variable

Summary The maximum number of lines of output that are printed during profiling. Package hcl Initial Value 100,000,000 Description \*default-profiler-limit\* is the maximum number of lines of output that are printed during profiling. The default value is large to ensure that you receive all possible output requested. \*default-profiler-limit\* only counts output lines for functions that are actually called during profiling. Therefore, if \*default-profiler-limit\* was set to, say, 19, and 20 functions were profiled, you would receive full output if one or more of the functions were not actually called during profiling.

### delete-advice

Function

Summary Removes a piece of advice.

Package hcl

Signature delete-advice dspec name => nil

dspec ::= fn-name | macro-name | (clos::method generic-fn-name [(class\*)])

Arguments	dspec	Specifies the functional definition to which the piece of advice belongs. The specifica- tion contains the name of the associated function. In the case of a method the list of classes is used to identify from which partic- ular method the advice should come. This list must correspond exactly with the classes corresponding to the specialized parameters for some method belonging to the generic function.
	name	A symbol naming the piece of advice to be removed. Since several pieces of advice may be attached to a single functional definition, the name is necessary to indicate which one is to be removed.
Values	delete-advice	e returns nil.
Description	advice. Advice Pieces of advice vice. They defit the function is instead of the f original definit functions, advi (in this case it is function).	a is the function used to remove a piece of is a way of altering the behavior of functions. e are associated with a function using defad- ine additional actions to be performed when invoked, or alternative code to be performed unction, which may or may not access the ion. As well as being attached to ordinary ce may be attached to methods and to macros s in fact associated with the macro's expansion
		that you need to quote the arguments.
Notes	delete-advice	is an extension to Common Lisp.
See also	remove-advice	

## dump-form

## Function

Summary	Dumps selected	d forms
Package	hcl	
Signature	dump-form form	stream => nil
Arguments	form	Form to be dumped.
	stream	Stream form is to be dumped on.
Values	Returns nil.	
Description		sed in conjunction with with-open-to-fasl- selected forms. See that entry for more details.
See also	dump-forms-to with-output-t	

## dump-forms-to-file

Summary	Dumps specifie	ed forms to a fasl file.
Package	hcl	
Signature	dump-forms-to-	-file pathname forms => nil
Arguments	pathname	Name of the fasl file to be created.
	forms	Forms to be dumped.
Values	Returns nil.	
Description	the Common L	o-file dumps specified forms to a fasl file. Use isp functions make-load-form and make- ring-slots to control the dumping of forms.

LispWorks uses the following naming conventions for fasl files, and it is recommended that you should use them too, to ensure correct operation of load and so on.

Table 9.1 Naming conventions for fasl files

Machine Architecture	Fasl File Extension
68000	.fasl
DECstation	.dfasl
SPARC	.wfasl
MIPS	.mfasl
Clipper	.cfasl
386 UNIX	.3fasl
386 Windows	.fsl

You can find the fasl file extension appropriate for your machine by looking at the variable system:\*binary-file-type\*. The variable system::\*binary-file-types\* contains a list of all the file extensions currently recognized by load.

For an example, see below.

		<pre>USER 1 &gt; (in-package "CL-USER") #<the 148="" 224="" 592="" 879="" cl-user="" external="" internal,="" package,=""> CL-USER 2 &gt; (defclass my-class () ((a :initarg :a :accessor my-a))) #<class my-class=""> CL-USER 3 &gt; (setq my-instance (make-instance 'my-class :a 23)) #<standard-instance my-class=""> CL-USER 4 &gt; (defmethod make-load-form ((self my-class))</standard-instance></class></the></pre>
Se	e also	with-output-to-fasl-file

## enlarge-static

Summary	Enlarges the siz	e of the first static segment.
Package	hcl	
Signature	enlarge-static	size => report
Arguments	size	The amount (in bytes) by which the static segment is to be enlarged.
Values	report	A report on the effect of the call to enlarge- static.
Description	allocate addition ments would car room, with argun and thus the size	n be used when the system would otherwise nal static segments (These additional seg- use the application to grow irreversibly.) Use ment ±, to find the size of the static segments, we by which to enlarge the first static segment. -up in multiples of 64K.

See also in-static-area room switch-static-allocation

## expand-generation-1

### Function

Summary	Controls expan	sion of generation 1.	
Package	hcl		
Signature	expand-generat	ion-1 on => size	
Arguments	on	If on is nil, generation 1 (the second genera- tion) is never expanded. If it is t, generation 1 is always expanded. If it is 1, generation 1 is only expanded if its current size is less than 500000 bytes. This is the default setting	1
Values	size	The current size of the image, in bytes.	
Description	insufficient spa	bsequent behavior of the garbage collector, if ce is freed by a mark-and-sweep. Depending a, the generation may be expanded, or the y be promoted.	
See also	clean-generat collect-gener collect-highe mark-and-swee set-gc-parame	ation-2 st-generation p	
extend-currer	extend-current-stack Function		

### Summary Extends the current stack.

Package hel

Signature	extend-current	-stack &optional how-much => size
Arguments	how-much	What percentage the stack should be extended by. The default is 50.
Values	size	The new size of the stack, after extending.
Description	Extend the curr	ent stack by the given percentage.
Example	To double the s	ize of the current stack:
	(hcl:ext	end-current-stack 100)

### extended-time

Macro

Summary	Produces useful on garbage colle	l timing information, including information ection activity.
Package	hcl	
Signature	extended-time	&body body => timing-info
Arguments	body	The forms to be timed.
Values	timing-info	Timing information on the forms contained in <i>body</i> .
Description	and system time generation 0. "I to promote from	nns show, respectively, total time, user time, e. "Main promote" indicates promotions from nternal promote" indicates when an attempt n one generation to the next causes promotion meration, to make room for the objects from

promotion process.

the lower generation. "fixup" is a part of the compaction and

Example	USER 22 > (hcl:extended-time (hcl:clean-generation-0)) user time = 1.320 system time = 0.100 Elapsed time = 0:00:01 Allocation = 144
	0 Page faults total gc activity = 0.960000 / 0.900000 / 0.060000
	<pre>main promote (0 calls) = 0.000000 / 0.000000 / 0.000000</pre>
	<pre>mark and sweep (200 calls) = 0.960000 / 0.900000 / 0.060000</pre>
	internal promote ( 100 calls = 0.000000 / 0.000000 / 0.000000
	promote ( 0 calls) = 0.000000 / 0.000000 / 0.000000
	fixup ( 100 calls) = 0.140000 / 0.120000 / 0.020000
	compact ( 0 calls) = 0.000000 / 0.000000 / 0.000000 NIL
	USER 23 >

See also time

## find-object-size

Summary	Returns the size in bytes of the representation of any Lisp object.
Package	hcl
Signature	find-object-size <i>object =&gt; size</i>
Arguments	<i>object</i> Any Common Lisp form.
Values	The result is an integer which is the number of bytes of heap storage currently used to represent the object. If the object takes up no heap storage (fixnum or character), then 0 is returned. Such objects are represented by an immediate value held in a single machine "word".

	The size of a heap object includes hidden space required to hold type and other information; for instance, a string of 10 characters occupies more than 10 bytes of storage.
Description	Certain Common Lisp objects are not represented by a single heap object; for instance, using find-object-size on a hash- table is misleading as the function returns the size of the hash-table descriptor, rather than the total of the descriptor and the hash-table-array. General vectors and arrays also have this property. All symbols are of the same size, since the print name is not part of a symbol object.
Example	USER 37 > (hcl:find-object-size (make-string 1000 :initial-element #\A)) 1012
See also	room total-allocation

## flag-not-special-free-action

Summary	Unflags an obje	ct for special action on garbage collection.
Package	hcl	
Signature	flag-not-speci	al-free-action <i>object</i> => nil
Arguments	object	The object on which the special actions are to be removed.
Values	Returns nil.	
Example		> (make-instance 'capi:title-pane) ANE "" 20F9898C>
	CL-USER 30 : 1 NIL	> (flag-not-special-free-action *)

See also add-special-free-action flag-special-free-action remove-special-free-action

## flag-special-free-action

### Function

Summary	Flags an object f	for special action on garbage collection.
Package	hcl	
Signature	flag-special-f:	ree-action <i>object</i> => t
Arguments	object	The object on which the special actions are to be performed. This cannot be a symbol.
Values	Returns t.	
Description		e current special-free-action functions are per- bject. Use flag-not-special-free-action ect.
Example		> (make-instance 'capi:title-pane) ANE "" 20F9898C>
	CL-USER 30 : 1 T	> (flag-special-free-action *)
See also	add-special-fr flag-not-speci remove-special	al-free-action

### gc-if-needed

Function

Summary Garbage collects if the previous call requires more space that is actually available.

Package hcl

Signature	gc-if-needed => nil
Arguments	None.
Values	Returns nil.
Description	This function checks to see if the amount of allocation from the previous call is more than system:*allocation-inter- val*, and if it is, performs a mark and sweep and promotion on generation 0. It also tries to reduce the big-chunk area. This is a fairly brief operation, and can be used whenever some operation is finished and may have left some garbage. The system itself uses it after compiling and loading files, when waiting for input, etc.
See also	avoid-gc get-gc-parameters mark-and-sweep normal-gc

# get-default-generation

Function

Summary	Returns the current default generation.
Package	hcl
Signature	<pre>get-default-generation =&gt; default-gen</pre>
Arguments	None.
Values	Returns the current default.

set-gc-parameters
without-interrupts
with-heavy-allocation

### The HCL Package

Description	By default, all new objects are allocated to a specific genera- tion. This function returns the current value of this default generation.
See also	allocation-in-gen-num clean-generation-0 collect-generation-2 collect-highest-generation expand-generation-1 set-default-generation *symbol-alloc-gen-num*

## get-working-directory

Summary	Finds the currer	nt working directory.
Package	hcl	
Signature	get-working-di	rectory => cwd
Arguments	None.	
Values	cwd	The current working directory, as a path- name.
Description	This function is used to find the current working directory. It returns a pathname, the directory component of which is the current working directory.	
Example	CL-USER 1 > (get-working-directory) #P"/nfs/watson/usr/users/neal/" 27	

## \*handle-existing-defpackage\*

Summary	-	Vorks' response when defpackage is used on kage that is different from the definition
Package	hcl	
Initial value	(:warn :modif	у)
Description	The standard explicitly declines to define what defpackage does if the named package already exists and is in a different state to that described by the defpackage form. The variable *handle-existing-defpackage* is an extension to Common Lisp which allows you to select between alternative behav- iors that are known to be useful. The two alternatives are to modify the package to conform exactly to the definition, removing features if necessary, or to merely add features specified in the defpackage but missing from the package. You can also control whether a condition is signalled. The variable consists of a list of any of the following:	
	:error	Signal an error.
	:warn	Signal a warning.
	:add	Add the new symbols to the externals, imports, and so on.
	:modify	Modify the package to have only these externals.
	:verbose	The signalled errors or warnings also con- tain details of the differences.
	The options :e	rror and :warn cannot be specified at the

The options :error and :warn cannot be specified at the same time. One of :add and :modify must be specified. Undistinguished internals (that is, internal symbols that are

	not imported or shadowed), <code>:intern</code> options and sizes are ignored when deciding whether to signal.
	Note that when you use :modify some symbols can be unin- terned if defpackage imports another symbol with the same name from another package through :import-from, :shad- owing-import-from or :export. This happens whether the symbol has a definition as a function, a variable, or nay other Lisp construct, so after making such a change in the package, you should re-execute the definitions that were (presumably erroneously) attached to the uninterned symbols.
Notes	*handle-existing-defpackage* is an extension to Common Lisp.
See also	defpackage

## \*max-trace-indent\*

### Variable

The maximum level of indentation used in trace output.
hcl
50
<pre>*max-trace-indent* is the maximum indentation that is used during output from tracing. Typically each successive invocation of tracing causes the output to be further indented, making it easier to see how the calls are nested. The value of *max-trace-indent* should be an integer.</pre>
<pre>USER 8 &gt; (setq hcl:*max-trace-indent* 4) 4 USER 9 &gt; (defun sum (n res) (if (= n 0)</pre>

	USER 10 > (trace sum) SUM
	USER 11 > (sum 3 0) 0 SUM > (3 0) 1 SUM > (2 0) 2 SUM > (1 0) 3 SUM > (0 0) 3 SUM < (0) 2 SUM < (1) 1 SUM < (3) 0 SUM < (6) 6
Notes	*max-trace-indent* is an extension to Common Lisp.
See also	trace

## normal-gc

Summary	Returns the image to normal garbage collection activity.
Package	hcl
Signature	normal-gc => t
Arguments	None.
Values	The function returns the single result t.
Description	normal-gc resets various internal parameters that determine the frequency and extent of garbage collection to their default settings.
	normal-gc is generally used in conjunction with avoid-gc, to cancel the effects of the latter.

#### The HCL Package

See also avoid-gc get-gc-parameters gc-if-needed mark-and-sweep set-gc-parameters without-interrupts with-heavy-allocation

### \*packages-for-warn-on-redefinition\* Variable

- Summary List of packages whose symbols should be checked for redefinition.
- Package hcl
- Initial Value nil
- Description Under some circumstances LispWorks warns you of redefinitions. \*packages-for-warn-on-redefinition\* is one of the variables which controls this behavior.

\*packages-for-warn-on-redefinition\* contains a list which specifies which packages should have their external symbols checked for redefinition.

The action taken by LispWorks on finding any redefinitions depends on the value of \*handle-warn-on-redefinition\*

See also \*handle-warn-on-redefinition\*

### print-profile-list

Summary	Prints a list of all symbols that have been profiled.
Package	hcl
Signature	print-profile-list &key sort limit => symbol-list

Arguments	sort	Determines which result is used to sort the list of profiled symbols. It may be one of three values :	
		<pre>:call — sort by the number of times each function was called.</pre>	
		:profile — sort by the number of times the function was found on the stack.	
		:top — sort by the number of times the function was found on the top of the stack.	
		If <i>sort</i> is omitted then the profiled symbols are printed as after the profiling run. The default is the value of the variable hcl:*default-profiler-limit*, which itself has default value :sort.	
	limit	The maximum number of symbols to print. The default is the value of the variable hcl:*default-profiler-sort*, which has the default value 100,000,000.	
Values	Returns a tabulated list of symbols, displayed within the parameters defined in any arguments given.		
Description	After profiling a Lisp form using the function profile, the symbols profiled may be printed out in order using print-profile-list. Thus, for example, the symbols found most often on the top of the stack can be quickly found. print-profile-list prints out a tabulated list of all the symbols which have been profiled showing how often each was called, profiled and found on the top of the stack.		
Example	First set up the profiler :		
	USER 4> (set-up-profiler :symbols '(car cdr maphash +))		
	USER 5> (profile (dotimes (a 100) (print (+ a a)) (print car '(foo))))		

Then call print-profile-list:

USER 6> (print-profile-list :sort :call) profile-stacks called 24 times Symbol called profile (%) top %) 100 1 (4) + 1 (4) CAR 100 0 ( 0) 0 ( 0) SYSTEM::DUMMY-STRUCTURE-ACCESSOR 12 0 ( 0) 0 ( 0) SYSTEM::DUMMY-STRUCTURE-SETTER 12 0 ( 0) 0 ( 0) MAPHASH 0 ( 0) 1 0 ( 0) Top of stack not monitored 96% of the time

Notes You can suppress printing of those symbols that are currently profiled but which were not called in the profiling run by setting the variable system::\*print-out-all\* to nil. The default value for \*print-out-all\* is nil. (This variable is loaded on demand by set-up-profiler.)

### profile

#### Macro

Summary	Runs the specifi	ed forms, and prints a performance profile.
Package	hcl	
Signature	profile &body	forms => final
Arguments	forms	The forms making up the program being profiled.
Values	final	The result of evaluating the final form.

Description	This macro starts up the LispWorks program profiler. This tool is useful for determining the time critical elements of a program.				
	At a regular time interval the Lisp process is halted and the execution stack is scanned for the presence of any symbols in the list *profile-symbol-list*. Counters are maintained for the number of calls to each symbol, the total number of times the symbol is found on the stack, and the number of times the profiler finds the symbol on the top of the stack.				
	This information is then presented as absolute numbers and as a percentage of the total number of calls to the profiler. These figures taken together give useful information about which functions the program spends most of its time execut- ing.				
Examples	<pre>USER 22 &gt; (set-up-profiler :symbols '(* gethash typep maphash)) NIL USER 23 &gt; (profile (let ((x 1))</pre>				
	profile-stacks called 12 times Symbol c (%)	alled	profile	e (%) top	
	MAPHASH 0 (0)*	1	0	(0)	
	0 (0)	50	1	(8)	
	SYSTEM::DUMMY-STRUCTURE-ACCESSON 0 (0)	R 6	0	(0)	
	SYSTEM::DUMMY-STRUCTURE-SETTER 0 (0)	9	0	(0)	
	TYPEP (8) 0 (0)	1	.9 1		
	GETHASH 3 (25) Top of stack not monitored 75% of	78 of the		(25)	
	30414093201713378043612608166064 000000000			6896051200	

See also	print-profile-list
	set-up-profiler

## \*profiler-threshold\*

### Variable

Summary	Controls which symbols are profiled on repeated profiling runs.
Package	hcl
Description	*profiler-threshold* is used with repeated profiling runs, to control which symbols are profiled. It is set by set-pro- filer-threshold.
See also	set-profiler-threshold

## \*profile-symbol-list\*

Variable

Summary	The list of symbols to be profiled.
Package	hcl
Description	*profile-symbol-list* is the list of symbols that are pro- filed if profile is called. Symbols in this list are monitored by the profiler to see if their function objects are on the stack when the profiler interrupts the Lisp process. The length of this list does not affect the speed of the profiling run.
Initial Value	nil
Notes	*profile-symbol-list* should normally be set by one of the above functions which check that the symbol is suitable for profiling before adding them to the list.

See also	add-symbol-profiler
	remove-symbol-profiler
	set-up-profiler

## remove-special-free-action

### Function

Summary	Removes the specified function from the special actions per- formed when flagged objects are garbage-collected.		
Package	hcl		
Signature	remove-special-free-action function => function-list		
Arguments	function	The function to be removed.	
Values	function-list	A list of the functions currently called to perform special actions, not including the one just removed.	
Description	Removes the specified function from the special actions per- formed when flagged objects are garbage-collected. (The spe- cial actions are added by add-special-free-action.)		
See also	add-special-free-action flag-special-free-action flag-not-special-free-action		

### remove-symbol-profiler

### Function

Summary Removes a symbol from the list of profiled symbols.

Package hcl

Signature remove-symbol-profiler symbol => nil

### The HCL Package

Arguments	symbol	A symbol to be removed from the *pro- file-symbol-list*.
Values	Returns mil.	
Description		-profiler removes a symbol from *profile- he list of profiled symbols.
See also	add-symbol-pro *profile-symbo	

## reset-profiler

Summary	Resets the profiler so that symbols below a given threshold are no longer profiled.	
Package	hcl	
Signature	reset-profiler	<pre>&amp;key according-to =&gt; nil</pre>
Arguments	according-to	One of two values — :profile or :top. This refers to which column of the profiling results reset-profiler uses to determine which symbols to delete from *profile- symbol-list*. The default is :profile.
Values	reset-profiler returns nil.	
Description	This function updates the list of symbols being profiled according to the results of the previous profiling run. reset- profiler runs down the list of symbols being profiled and removes any symbols whose appearance in the previous pro- filing run falls below the value *profiler-threshold*. In this way the number of symbols being considered by the pro- filer can be reduced to just those which are important.	

Example	(reset-profiler :according-to :top)
Notes	Reducing the number of symbols in profile-symbol-list does not actually speed up the execution of the form being profiled, but does reduce the setting up time of the profiler and the size of the list of results.
See also	profile print-profile-list set-profiler-threshold
save-image	Function
Summary	Saves the image to a new file.
Package	hcl
Signature	<pre>save-image filename &amp;key gc type normal-gc restart-function console environment remarks clean-down =&gt; nil</pre>

Arguments	filename	A string. It is the name of the file that the image is saved as. This name should not be the same as the original name of the image.
	gc	If non-mil, there is a garbage collection before the image is saved. The default value is t.
	type	Determines if some global variables are cleared before the image is saved. You can generally use the default value, which is :user.
	normal-gc	If this is $t$ the function normal-gc is called before the image is saved. The default is $t$ .

	restart-function	Specifies a function (with no arguments) to be called when the image is started. The default is nil. The restart-function is called after the init-file is loaded.
	console	<i>console</i> controls whether the new image will be a Console or GUI application and when, if ever, to make a console window in the lat- ter case. Possible values are discussed below.
	environment	<i>environment</i> controls whether the LispWorks environment is started on restart. Possible values are discussed below.
	remarks	<i>remarks</i> adds a comment to the save history. The value should be a string.
	clean-down	When t, calls (clean-down t).
Values	Returns nil.	
Description	Using this function, you can make modifications to the image, and then save the image with the modifications inside. This is particularly important when patches are sent by The Harlequin Group Limited. Do not use this function when the programming environ- ment is running. Instead, create an initialization script. The possible values for <i>console</i> are as follows:	
	:default	Unchanged since previous save.
	t	A Console application is saved, else a Win- dows application is saved which create its own console according to the other possible values.

:input, :output,	:io
------------------	-----

	Whenever input, output or any io is attempted on <b>*terminal-io</b> *.
:init	At startup, if input and output are not redirected
:always	At startup, even if input and output are redirected.

The LispWorks image shipped by Harlequin has it set to :input.

The possible values for *environment* are as follows:

:default	Unchanged since previous save.
nil	Start with just the TTY listener.
t	Start the environment automatically, no TTY
	listener.

#### :with-tty-listener

Start the environment automatically, but still have a TTY listener.

The LispWorks image shipped by Harlequin has it set to t.

You should not try to save a new image over an existing one on SPARC based machines. Always save images using a unique image name, and then, if necessary, replace the new image with the old one after the call to save-image has returned.

Example Here is an example initialization script. Save this to a file such as my-file.lisp, then run LispWorks with the command line argument -init c:/my-file.lisp to build the image my-image.exe.

```
(load "my-application")
(save-image "my-image")
(quit)
```

## set-default-generation

Summary	Set the current generation for storage allocation.
Package	hcl
Signature	<pre>set-default-generation num =&gt; num</pre>
Arguments	<i>num</i> The number of generation from which to do future allocation.
Values	Returns <i>num</i> .
Description	Set the current generation for storage allocation. By default the system allocates memory from the youngest generation (generation 0).
Examples	<pre>(set-default-generation 1)     ;; allocate from an     ;; older generation (set-default-generation 0)     ;; return to normal</pre>
See also	allocation-in-gen-num clean-generation-0 collect-generation-2 collect-highest-generation expand-generation-1 get-default-generation set-promotion-count *symbol-alloc-gen-num*

## set-gc-parameters

Summary	Sets the parameters from the garbage collector.	
Package	hcl	
Signature	set-gc-paramet	ers &key <i>keywords</i> => <no values=""></no>
Arguments	The keywords for this function are:	
	big-object	An object that is bigger than this value is "big", i.e. is not allocated from the small objects buffer, but from the big-chunk area (if it is allocated in generation 0 in the nor- mal way)
	enlarge-by-segments	
		A minimum for how much the image grows each time a segment is enlarged, as a multi- ple of 64K. This parameter is ignored when adding a static segment.
	maximum-buffer	-size
		Maximum size of the small objects buffer.
	minimum-buffer-size	
		Minimum size of the small objects buffer.
	maximum-overflow	
		Maximum size of the small-objects buffer in the big-chunk area.
	minimum-overflow	
		Minimum size of the small-objects buffer in the big-chunk area.

#### minimum-for-promote

Controls the frequency of promotions. Setting :minimum-for-promote to a high value causes the system to promote less frequently. This may improve performance for programs that allocate a lot of data for a short term and then delete it.

#### minimum-for-sweep

Controls when a mark-and-sweep takes place. Setting minimum-for-sweep to a high value causes the system to mark and sweep less often, which means it has to grow. The CPU time spent in garbage collection is mostly smaller, but the process is bigger and may cause more disk access.

#### new-generation-size

In conjunction with the function expandsecond-generation, :new-generationsize controls the expansion of generations. If :new-generation-size is 0, the generation is not expanded. Otherwise, the generation is expanded by :new-generation-size or by the amount of space needed, whichever is the greater.

#### promote-max-buffer and promote-min-buffer

During promotion, a buffer is allocated in the generation being promoted into, and the objects promoted are moved into it. These keywords control the size of this buffer.

Values

None.

Description This function sets the parameters of the garbage collector, using the keywords described above.

See also get-gc-parameters

## set-minimum-free-space

Summary	Sets the minimum free space for a segment of the specified generation.	
Package	hcl	
Signature	<pre>set-minimum-free-space gen-num size &amp;optional segment =&gt; generation-size</pre>	
Arguments	gen-num	The generation to be affected.
	size	The size (in bytes) to set the segment to.
		The <b>:segment</b> keyword allows you to spec- ify the segment to be affected (the default is the first segment of the generation).
Values	generation-size	A list showing information for the genera- tion just specified in the call.
Description	Sets the minimum free space for a segment of the specified generation. By default, affects the first segment — use the :segment keyword to affect a different segment of the generation. The minimum free space is shown by room.	
See also	clean-generation-0 collect-generation-2 collect-highest-generation expand-generation-1 room set-promotion-count	

## set-profiler-threshold

Function

Summary	Sets the percentage threshold for symbols to be profiled in a subsequent run.
Package	hcl
Signature	<pre>set-profiler-threshold value =&gt; value</pre>
Arguments	<i>value</i> must be a fixnum between 0 and 100.
Values	set-profiler-threshold returns value.
Description	This function sets the value of *profiler-threshold* below which symbols are not profiled in a repeated profiling run. After a profiling run, all the symbols being profiled have a percentage value for the amount of time they were on the top of the stack. If *profiler-threshold* is set to 40 then by running reset-profiler with argument :top all symbols which are found on the top of the stack less than forty per- cent of the time are removed from the list of those symbols considered for profiling.
Example	(set-profiler-threshold 40)
See also	reset-profiler profile *profiler-threshold*

### set-promotion-count

Function

Summary Controls when objects can be promoted to the next generation.

Package hcl

Signature	<pre>set-promotion-count gen-num count &amp;optional segment =&gt; count</pre>	
Arguments	gen-num	The generation number affected.
	count	The number of garbage collections survived by objects in that generation, before promo- tion. If count is nil, the function returns the current promotion count setting.
	segment	Specifies which segment of the generation is to be affected. The default is the lowest seg- ment of the generation.
Values	Returns <i>count</i> .	
Description	Controls how many garbage collections an object in a seg- ment must survive before promotion to the next generation.	
See also	clean-generation-0 collect-generation-2 collect-highest-generation expand-generation-1	

## set-up-profiler

Summary	Declares the par	rameter values of the profiling function.
Package	hcl	
Signature	set-up-profile	r &key symbols packages interval
Arguments	symbols	A symbol or a list of symbols. Each symbol is checked to see if it is suitable for profiling and if so it is added to the list *profile- symbol-list*.

	packages	A valid package name or list of package names, or :all, meaning all packages. All the symbols in the packages are checked as above. If a :symbols keyword is present then this argument is ignored.
	interval	A positive integer representing time in microseconds. The default value is 10000.
Values	The time interval is returned.	
Description	<ul> <li>set-up-profiler is used to declare the values of the parameters of the profiling function. Three values are required:</li> <li>The symbols or packages to be monitored by the profi</li> </ul>	
	• The time interval between profiling samples. This time interval is in microseconds, and the minimum value is 10000, i.e. 10 ms.	
Example	(set-up-profil	er :symbols '(car cdr) :interval 50000)
See also	add-symbol-pro profile remove-symbol·	

## source-debugging-on-p

Summary	Tests if source le	evel debugging is on for compiled code.
Package	hcl	
Signature	<pre>source-debugging-on-p =&gt; bool</pre>	
Arguments	None.	
Values	bool	If t, source level debugging is on.

Description	Returns t if source level debugging is on for compiled code; otherwise returns nil.
See also	toggle-source-debugging

### switch-static-allocation

### Function

Summary	Controls whether objects are allocated in the static area.	
Package	hcl	
Signature	<pre>switch-static-allocation flag =&gt; previous-flag</pre>	
Arguments	flag	If <i>flag</i> is non-nil, subsequent objects are allocated in the static area; if <i>flag</i> has any other value, objects are allocated conventionally.
Values	switch-statio	c-allocation returns the previous setting of
Description	Objects in the static area are garbage-collected, but not moved.	
	You should avoid using this function.	
See also	enlarge-stat: in-static-are	

## \*symbol-alloc-gen-num\*

Variable

- Summary Specifies the generation to which interned symbols and their symbol names are allocated.
- Package hcl

### The HCL Package

Initial Value	2
See also	allocation-in-gen-num get-default-generation set-default-generation

## toggle-source-debugging

Summary	Toggles the production of source level debugging informa- tion.	
Package	hcl	
Signature	toggle-source-debugging &optional on => bool	
Arguments	on	Flag (t or nil) to control the resulting set- ting of the variables. The default is t.
Values	bool	The current state of source level debugging: t if source level debugging is on.
Description	toggle-source-debugging controls the setting of six com- piler parameters. (These parameters are initially set in the configuration file configure.lisp.) For all these parameters, the setting nil improves the compilation speed.	
	level debugging function definit	s relate to information required for source g, and other operations that need a record of cions (operations such as finding all changed inding the source of a definition).
	The parameters (all internal to the compiler package) are:	
	*produce-xref-info*	
		When t, the compiler produces information for the Cross Referencer.

#### \*load-xref-info\*

When t, the cross-referencing information produced by the compiler is loaded when the corresponding file is loaded.

#### \*notice-changed-definitions\*

When t, the Cross Referencer notices when a definition is changed interpretively.

#### \*generate-source-recording\*

When t, the compiler generates source recording information.

```
*source-file-recording*
```

When t, the source file recording information produced by the compiler is loaded with a fasl file or text file.

```
*source-level-debugging*
```

When t, the compiler generates information used by the debugger.

toggle-source-debugging modifies the status of the variables, and then returns the new value. To test the setting of any variable, without modifying it, use source-debuggingon-p.

See also source-debugging-on-p

## total-allocation

#### Function

Summary Calculate memory consumed since the image was started.

Package hcl

Signature total-allocation

#### The HCL Package

Arguments	None.
Values	Returns the amount allocated
Description	This function calculates the total amount of memory con- sumed since the current image was created. Use at the start and end of a piece of code, to see how much it allocates.
See also	find-object-size room

## \*traced-arglist\*

- Summary The list of arguments given to the function being traced.
- Package hcl
- Initial Value nil
- Description Upon entering a function that is being traced, \*tracedarglist\* is bound to the list of arguments given to the function. \*traced-arglist\* is then printed after the function name in the output from tracing. It is accessible in the :before and :after forms to trace. However care should be used when manipulating this variable, since it is the value of \*traced-arglist\* itself that is used when calling the traced function. Thus if this value is altered by the :before forms then the function receives the altered argument list.

Example	USER 14 > (trace	(+ :before
		((setq *traced-arglist*
		(mapcar #'1+
		<pre>*traced-arglist*)))))</pre>
	+	
	USER 15 > (+ 1 2	3)

	0 + > (1 2 3) (2 3 4) 0 + < (9) 9
Notes	*traced-arglist* is an extension to Common Lisp.
See also	trace

## \*traced-results\*

Summary	The list of results from the function being traced.		
Package	hcl		
Initial Value	nil		
Description	Upon leaving a function that is being traced, *traced- results* is bound to the list of results from the function. *traced-results* is then printed after the function name in the output from tracing. It is accessible in the :after forms to trace. However care should be used when manipulating this variable, since it is the value of *traced-results* itself that is used when returning from the traced function. Thus if this value is altered by the :after forms then the caller of the traced function receives the altered results.		
Example	USER 5 > (trace (ceiling :after ((setq *traced-results* (mapcar #'1- *traced-results*)))))		
	CEILING USER 6 > (multiple-value-call #'+ (ceiling 4 3))		
	0 CEILING > (4 3) 0 CEILING < (2 -2) (1 -3) -2		

Notes *trac	ed-results* is an	extension to	Common Lisp.
-------------	-------------------	--------------	--------------

See also trace

## \*trace-indent-width\*

Summary	The amount of extra indentation in the trace output for each level of nesting.		
Package	hcl		
Initial Value	2		
Description	*trace-indent-width* is the extra amount by which the traced output for function calls is indented upon entering a deeper level of nesting (i.e. a traced call from a function that is itself traced). If it is 0 then no indentation occurs.		
Example	USER 7 > (setq *trace-indent-width* 4 *max-trace-indent* 50)		
	50 USER 8 > (defun quad (a b c) (- (* b b) (* 4 a c)))		
	QUAD USER 9 > (trace quad *)		
	* USER 10 > (quad 4 3 14)		
	0 QUAD > (4 3 14) 1 * > (3 3) 1 * < (9) 1 * > (4 4 14) 1 * < (224) 0 QUAD < (-215) -215		
Notes	*trace-indent-width* is an extension to Common Lisp.		
See also	trace		

## \*trace-level\*

## Variable

Summary	The current depth of tracing.		
Package	hcl		
Initial Value	0		
Description	*trace-level* is a special variable whose value is the cur- rent depth of tracing. The current value of *trace-level* is printed before the function name during the output from tracing.		
Example	USER 8 > (defun fac (n) (if (<= n 1) 1 (* n (fac (1- n)))))		
	FAC USER 9 > (trace fac)		
	FAC USER 10 > (fac 3)		
	0 FAC > (3) 1 FAC > (2) 2 FAC > (1) 2 FAC < (1) 1 FAC < (2) 0 FAC < (6) 6		
Notes	<pre>*trace-level* is an extension to Common Lisp.</pre>		
See also	trace		

# \*trace-print-circle\*

Summary	Controls how circular structure are printed in trace output.
Package	hcl

The HCL Package

Initial Value nil Description \*trace-print-circle\* controls how circular structures are printed during output from tracing. It allows the printing of circular structures by the tracer to be controlled independently of the usual printing mechanism, which is governed by \*print-circle\*. \*print-circle\* is bound to the value of \*trace-print-circle\* while printing tracing information. Example USER 19 > (setq \*trace-print-circle\* t) т USER 20 > (defun circ (1))(rplacd (last 1) 1) 1) CIRC USER 21 > (trace second) SECOND USER 22 > (second (circ '(1 2 3 4)))  $0 \text{ SECOND} > (\#1=(1 \ 2 \ 3 \ 4 \ . \ \#1\#))$ 0 SECOND < (2) 2Notes \*trace-print-circle\* is an extension to Common Lisp. See also trace

## \*trace-print-length\*

Summary	The number of components of an object that are printed in trace output.
Package	hcl
Initial Value	100
Description	*trace-print-length* controls the number of components of an object which are printed during output from tracing. If

its value is a positive integer then the first \*trace-printlength\* components are printed.

\*print-length\* is bound to the value of \*trace-printlength\* while printing tracing information. If \*traceprint-length\* is nil then all the components of the object are printed.

Example USER 5 > (trace append) APPEND USER 6 > (setq \*trace-print-length\* 3) 3 USER 7 > (dotimes (i 10) (setq li (if (zerop i) nil (cons i li)))) NIL USER 8 > (append li '(a b))  $0 \text{ APPEND} > ((9 \ 8 \ 7 \ \dots) \ (A \ B))$  $0 \text{ APPEND} < ((9 \ 8 \ 7 \ \ldots))$ (987654321AB) Notes \*trace-print-length\* is an extension to Common Lisp. See also trace

## \*trace-print-level\*

Summary	The depth to which nested objects are printed in trace output.		
Package	hcl		
Initial value	5		
Description	*trace-print-level* controls the depth to which nested objects are printed during output from tracing. If its value is a positive integer then components at or above that level are suppressed. By definition an object to be printed is consid-		

	ered to be at level 0, its components are at level 1, their sub- components are at level 2, and so on.	
	*print-level* is bound to the value of *trace-print- level* while printing tracing information. If *trace-pri level* is nil then objects are printed without regard to depth.	
Examples USER 8 > (trace append)		
	APPEND USER 9 > (dotimes (i 10) (setq li (if (zerop i) nil (list i li))))	
	NIL USER 10 > (append li '(a b)) O APPEND > ((9 (8 (7 (6 #)))) (A B)) O APPEND < ((9 (8 (7 (6 #))) A B)) (9 (8 (7 (6 (5 (4 (3 (2 (1 NIL))))))) A B)	
Notes	*trace-print-level* is an extension to Common Lisp.	
See also	trace	

## \*trace-print-pretty\*

- Summary Controls the amount of whitespace in trace output.
- Package hcl
- Initial Value nil
- Description \*trace-print-pretty\* controls the amount of whitespace printed during output from tracing. If it is not nil then extra whitespace is inserted to make the output more comprehensible. \*print-pretty\* is bound to the value of \*traceprint-pretty\* while printing tracing information.
- Examples USER 6 > (trace macroexpand-1)

```
MACROEXPAND-1
              USER 7 > (setq *trace-print-pretty* t
                           *print-pretty* nil)
              NIL
              USER 8 > (defmacro sum (n))
                           `(do ((i 0 (1+ i)))
                                 (res 0 (+ i res)))
                                ((= i ,n) res)))
              SUM
              USER 9 > (macroexpand-1 '(sum 3))
              0 \text{ MACROEXPAND-1} > ((SUM 3))
              0 \text{ MACROEXPAND-1} < ((DO ((I 0 (1+ I))))
                                     (RES 0 (+ I RES)))
                                   ((= I 3)
                                   RES))
                                 T)
              (DO ((I 0 (1+ I)) (RES 0 (+ I RES))) ((= I 3) RES))
              т
Notes
              *trace-print-pretty* is an extension to Common Lisp.
See also
              trace
```

who-calls

Summary	Returns a list of the callers of function.	
Package	hcl	
Signature	who-calls function => callers	
Arguments	function	Function whose callers are required.
Values	callers	A list of callers of the function.
Description	who-calls returns a list of the callers of function. See also the Editor commands List Callers and show Paths To.	

See also calls-who

# with-heavy-allocation

## Macro

Summary	Slows up garbag that allocates a l	ge collection during the execution of code ot of space.
Package	hcl	
Signature	with-heavy-all	ocation &rest <i>body</i> => <i>result</i>
Arguments	body	The forms for which you want the garbage collector to behave differently from normal.
Values	result	The result of executing <i>body</i> .
Description	but is not intera ried out less free	r use with code that allocates a lot of space ctive. It ensures that garbage collection is car- quently while these forms are being executed ge collection may take longer).
See also	avoid-gc gc-if-needed get-gc-paramet mark-and-sweep normal-gc set-gc-paramet without-intern	ers

# with-output-to-fasl-file

Summary	Sends output to a fasl file on disk.
Package	hcl
Signature	with-output-to-fasl-file (stream pathname &rest options) &body body => nil

Arguments	stream	Stream corresponding to the fasl file to be created.
	pathname	Name of the fasl file to be created.
	body	Forms, some of which may be dumped.
Values	Returns nil.	
Description		o-fasl-file is used in conjunction with ump selected forms.
	files, and it is re	s the following naming conventions for fasl commended that you should use them too, to operation of load and so on.

\_

Table 9.2 Naming conventions for FASL files

Machine Architecture	Fasl File Extension
Alpha	.afasl
RS/6000	.rfasl
HPPA	.pfasl
DECstation	.dfasl
SPARC	.wfasl
Clipper	.cfasl
Other PC	.fsl

You can find the fasl file extension appropriate for your machine by looking at the variable system:\*binary-file-type\*. The variable system::\*binary-file-types\* contains a list of all the file extensions currently recognized by load.

#### The HCL Package

Example SYSTEM 5 > (with-output-to-fasl-file (s "/tmp/foo.fasl") (dump-form '(print 'hello) s)) 0 NIL SYSTEM 6 > (load "/tmp/foo.fasl") ; Loading fasl file /tmp/foo.fasl HELLO #P"/tmp/foo.wfasl" See also dump-form dump-forms-to-file

# 10

# The LINK-LOAD Package

This chapter describes the symbols in the link-load package.

## break-on-unresolved-functions

Package	link-load	
Syntax	break-on-unres	olved-functions & optional stream
Arguments	stream	An output stream for message reporting. If set to nil, then no output will be produced. By default this is t.
Description	break-on-entry enced (that is, u undefined forei code, a Lisp err	unresolved-functions function produces code for all currently undefined but refer- unresolved) foreign symbols, so that if an gn function is called from within the foreign or will occur. Break-on-entry code will also be ny new unresolved symbols loaded later in on.

The special variable foreign:\*break-on-unresolvedfunctions\* will, when set to non-nil, produce break-onentry code for all new unresolved symbols that are loaded, but won't do so for symbols already loaded. By default this variable is set to nil.

See Also read-foreign-modules

## foreign-symbol-address

Package	link-load	
Syntax	foreign-symbol => <i>result</i>	l-address name &key (errorp t) (functionp t)
Arguments	name	The name of a foreign symbol.
	errorp	A boolean.
	functionp	A boolean.
Values	result	The address of <i>name</i> or nil.
Description	whether a fore foreign-symbo loaded into the	ymbol-address function is used to find out ign symbol is defined, by looking for it in the l table. If its associated object code has been e image, its address is returned. Otherwise nil less <i>errorp</i> is nil.
	when a symbol the default val	word defines the behavior of the function l has not been defined. If it is non-ni1 (which is ue), then an error will be signalled. If it is ni1, e reported, and the function will return ni1.
	sought. If it is a name is the name	teyword is used to specify the kind of symbol t, foreign-symbol-address will assume that ne of a function. If it is nil it will assume that ne of a variable. The default value is t.

Example (foreign-symbol-address 'chmod)

See also get-foreign-symbol

# get-foreign-symbol

Package	link-load	
Syntax	get-foreign-syn	nbol name &optional force => result
Arguments	name force	A symbol or string. A keyword.
Values	result	A foreign symbol.
Description		ets a foreign symbol or it may be used to er an undefined symbol.
	symbol. If it is a the function lis	l or string to look up or to create as a foreign symbol, the symbol looked for is that which gp-name-to-foreign-name would produce. If it is taken literally
	symbol, <i>force</i> for provides a refer	the symbol is not already defined as a foreign ces it to be an undefined foreign symbol. This ence to the symbol so that a subsequent call n-modules will attempt to resolve it
Example	(get-foreign-s	wmbol 'my-func-not-yet-loaded t)
Notes	examine whether symbol-address fli:define-for	necessary to use this function.In order to er a foreign symbol is defined, use foreign- s. The act of defining a foreign function using reign-function makes the symbol unde- e of force is not usually needed.

See also foreign-symbol-address lisp-name-to-foreign-name read-foreign-modules

# lisp-name-to-foreign-name

Function

Package	link-load	
Syntax	lisp-name-to-f	oreign-name name &key language
Arguments	name	A symbol representing a Lisp name. (Strings are passed unchanged through the func- tion.)
	language	If :c then an equivalent 'C' name is pro- duced. :FORTRAN is an alternative.
Description	-	rovides an equivalent foreign name for a Lisp ng on the keyword language.
Values	name supplied. string unchange replaces occurre score. The Fortr	rned which is a foreign equivalent of the Lisp If name is a string, the function returns the ed. If language is a symbol, the 'C' version ences of '-' with '_' and adds a leading under- an version replaces occurrences of '-' with eading and trailing underscore.
Example	(lisp-name-to- "_lisp_name_wi	foreign-name 'lisp-name-with-hyphens) th_hyphens"
See Also	get-foreign-s	ymbol

# process-foreign-code

Macro

Package link-load

Syntax	process-foreign-code <i>c-string</i> &key <i>language control</i> =>	
Arguments	c-string	A string.
	language	One of :c, :c++ or :ansi-c.
	control	One of :fasl, :object, or :source.
Description	file. process-fo	ws C or C++ code to be included in a Lisp fasl preign-code is emitted by process-foreign- sing C++ header files.
	<i>c-string</i> is a strin	ng containing C or C++ source code.
	<i>language</i> is eithe	er :c, :c++ or :ansi-c, and is :c++ by default.
	default. If <b>:fas</b> : into a temporar in the fasl file. I compile time in and loaded into	<pre>:fas1, :object Or :source. It is :fas1 by L, the foreign code is compiled at compile time y .o file. The resulting object module is stored f :object, the foreign code is compiled at to a .o file with the same name as the lisp file, o the image when the fas1 file is loaded. If lation and loading of the foreign code is done le is loaded.</pre>
Example	contr: ; When the lis:	<pre>gn-code "int sum (int a, int b)         { return a+b; } " fol :source) gp file containing this statement is c-code is compiled and also loaded into</pre>
See Also	process-forei	gn-file

## process-foreign-file

- Package link-load
- Syntax process-foreign-file files &key dff language preprocess preprocessor preprocessor-options virtual-functions accessors casesensitive c-code-control resolve-errors =>

The LINK-LOAD Package

Arguments	files	One or more filenames.
	dff	A filename.
	language	One of :c++, :g++, :arm, or :any-c++.
	preprocess	A boolean.
	preprocessor	A string.
	preprocessor-opti	ions
		A string.
	virtual-functions	s A boolean.
	accessors	A boolean.
	case-sensitive	See description.
	c-code-control	One of :fasl, :object, or :source.
	resolve-errors	A boolean.
Description	-	preign-file function takes a file or files of the sector o

The process-foreign-file function takes a file or files of foreign declarations — usually header files — and parses them, producing 'dff' files of Lisp calls to define-foreignfunction, define-foreign-variable, define-foreigntype, and so on, that will, when compiled, produce a Lisp interface to the foreign code.

> process-foreign-file will use a preprocessor program specified by the variable system::\*preprocessor\*. The default value is /lib/cpp.

*files* gives the name of the header files or file to be processed. The name of a file consists of *source-file-name* and *source-file-type* (typically .h).

*dff* is an output file which will contain the Lisp foreign function definitions. The default value is nil, in which case the dff file will be *source-file-name-dff.lisp*. (See *files*, above.)

language specifies the language the header files are written in. This may be one of :c++, :g++ (both GNU g++ compiler-compatible), :arm (ARM C++/AT&T/Lucid compatible), :c (standard C header files), and :any-c++ (any C++ variety). By default this is :any-c++.

*preprocess*, when non-nil, runs the preprocessor on the input files. The default value is t.

preprocessor is a string containing the pathname of the preprocessor program. By default this is the value of ccl:\*preprocessor\*.

preprocessor-options is a string containing command line options to be passed to the preprocessor if it is called. By default this is the value of ccl:\*preprocessor-options\*.

virtual-functions, when non-nil, will assume that all C++ member functions are virtual. This creates separate caller stubs for each function. You will need to set this to t if you want to interface to any form of virtual function. This option will also enable interfacing to inline functions. LispWorks does not attempt to interface to the internals of virtual method dispatching. By default this is t.

accessors, if t, ensures accessors are created for all class slots. If nil, neither reader nor writer functions are created. The default value is t.

*case-sensitive* specifies whether to maintain case sensitivity in symbol names as in the source files. Values can be:

- t the names of all Lisp functions and classes created are of the form | name |.
- nil all foreign names are converted to uppercase and an error is signalled if any name clashes occur as a result of this conversion. For example, Onetwothree becomes ONETWOTHREE.

<ul> <li>:prefix — changes lowercase to uppercase and concat nates the string with the string held in sys:*prefix-name-string*. For example, OneTwOTHREE becomes FOR EIGN-ONETWOTHREE.</li> <li>(list :user-routine function-name) — enables you to pass your own function for name formatting. Your function</li> </ul>	
tion must take a string argument and return a string result. It is not advised to use destructive functions (e.g	)-
If <i>case-sensitive</i> takes any other value, names are not change The default value is <b>*case-sensitive</b> *.	d.
<i>c-code-control</i> can be either <b>:fas1</b> , <b>:object</b> Or <b>:source.</b> The default value is <b>*c-code-contro1*</b> . This control keyword is used when emitting the <b>process-cpp-code</b> definition.	5
<i>resolve-errors</i> , when non-nil, creates null structures for all undefined foreign datatypes. The default value is nil.	
Example (process-cpp-file "test.h" :accessors t :case-sensitive nil)	э
; This creates the dff-file test-dff.lisp, which ; contains the necessary foreign definitions for ; declaration in test.h.	
<pre>(compile-file "test-dff" :load t) ;; compile and load test-dff.lisp</pre>	
<pre>(read-foreign-modules "test.o") ;; load original object module</pre>	
See Also process-foreign-code read-foreign-modules	

# read-foreign-modules

Package	link-load
Syntax	read-foreign-modules &rest module-names => nil
Arguments	<i>module-names</i> A sequence of strings or pathnames.
Values	nil
Description	The read-foreign-modules function reads object files of var- ious formats into the Lisp image. Unresolved references are resolved wherever possible and the names of the foreign functions are made available to the Lisp for direct calling from the Lisp if desired. With no argument, read-foreign- modules scans the default libraries looking for definitions of referenced but undefined symbols.
	The <i>module-names</i> argument is a sequence of items represent- ing object files to be loaded. The items may be of type string or pathname, and will be used to look up a corresponding file in the file system. The only exception is if an item is a string beginning "-1" in which case the rest of the string is used to look up a library file using the format strings in the variable *default-library-name-formats*. Object files of various formats and library files can be handled by read-foreign- modules.
Example	<pre>(read-foreign-modules "/usr/users/clc/projects/head.o"</pre>
Notes	The read-foreign-modules function actually adds the module-names to the list of modules in the variable *default-libraries* and then tries to resolve any unde- fined symbols using this list. The function get-foreign-

symbol may be called to explicitly force a symbol onto the undefined list or the act of defining a foreign function (fli:define-foreign-function) will do it implicitly.

read-foreign-modules may be called at any time during the running of a program and a particular object file may be loaded as often as is necessary.

A warning of any new unresolved references will be printed out after the reading has finished if the flag \*unresolvedmessages\* is set to t (the default is nil). By default messages are printed out about which object modules are being loaded. This may be switched off by setting \*coff-loadingverbose\* to nil.

See Also get-foreign-symbol

# The LISPWORKS Package

This chapter describes symbols available in the LISPWORKS package. This package is used by default when LispWorks is started.

## 8-bit-string

Type

Summary	The 8 bit string	type.
Package	lispworks	
Signature	8-bit-string $l$	ength
Arguments	length	The length of the string (or *, meaning any).
Description	01	ngs that can hold simple chars of codes he string type that is guaranteed to always lement.

# 16-bit-string

Summary	The 16 bit string	g type.
Package	lispworks	
Signature	16-bit-string	length
Arguments	length	The length of the string (or *, meaning any).
Description	The type of strings that can hold simple chars of codes 065533. This is the string type that is guaranteed to always take 16 bits per element.	

Туре

Macro

appendf
---------

Summary	Apponde liste t	o the end of a given list.
Summary	Appends lists t	o the end of a given list.
Package	lispworks	
Signature	appendf place &	rest lists => result
Arguments	place	A place.
	lists	A set of lists.
Values	result	An object.
Description	0	cro appendf appends the lists given by <i>lists</i> to ist in <i>place</i> . See append for more details.
See also	removef	
oase-char		Туре

# base-char

Summary	The base character	type.
		-J <b>F</b>

Package	lispworks
Signature	base-char
Description	The type of base characters. A synonym for base-character, but with standard spelling.
See also	base-character

## base-character

## Type

- Summary The base character type.
- Package lispworks
- Signature base-character
- Description The type of base characters.

# base-character-p

Summary	Tests if an objec	ct is a base character
Package	lispworks	
Signature	base-character	с-р object => bool
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is a base character; nil otherwise.
Description	This is the predicate for base characters.	
See also	base-characte	r

# base-char-p

Summary	Tests if an object is a base character	
Package	lispworks	
Signature	base-char-p object => bool	
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is a base character; nil otherwise.
Description	This is also the predicate for base characters, only with stan- dard spelling.	
See also	base-char base-character	<b>-</b> p

## base-char-code-limit

### Constant

Function

Function

Summary	Upper bound for character codes in base characters.
Package	lispworks
Description	The upper exclusive bound for values of (char-code char) among base characters.
See also	base-char

## base-string-p

- Summary Tests if an object is a base string.
- Package lispworks

Signature	<pre>base-string-p object =&gt; bool</pre>	
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is a base string; nil otherwise.
Description	This is the predicate for base strings.	
See also	base-string	

## call-next-advice

Summary	Calls the next piece of advice associated with a function.		
Package	lispworks		
Signature	call-next-advice args		
Arguments	<i>args</i> are arguments to be given to the next piece of advice to be called. Any number of arguments may be given in this way, including keyword arguments, and there is no require- ment for pieces of around advice to receive the same number of arguments as the original definition expected.		
Values	call-next-advice returns the values produced by the call to the next piece of advice (or to the combination of before and after advice and the original definition).		
Description	call-next-advice is the local function used to invoke the next item in the ordering of pieces of advice associated with a function. It can only be called from within the scope of the around advice. Advice may be attached to a function by defadvice and this allows the behavior of a function to be modified. Extra code to be performed before or after the function may be simply added by creating before or after advice for it. Around advice is more powerful and replaces		

the original definition. All the advice for a function is ordered with the around advice coming first.

The first piece of around advice receives the arguments to the function and may return any values at all. It has access to the rest of the advice, and to the original definition, by means of call-next-advice. A call to this from within the body of the around advice invokes the next piece of around advice with the arguments given to call-next-advice. The last piece of around advice in the ordering invokes the sequence of before advice, the original definition, and after advice if it calls call-next-advice. Around advice may contain any number of calls to call-next-advice, including no calls.

Notes call-next-advice is an extension to Common Lisp. See the LispWorks User Guide for a broader discussion of advice.

#### compile-system

Summary	The function compile-system compiles all the files in a system necessary to make a consistent set of object files.	
Package	lispworks	
Signature	compile-system target-directory	system-name &key force simulate load args
Arguments	system-name	A symbol representing the name of the system. The system must have been defined already using the defsystem macro.
	force	If t then all the files in the system are com- piled regardless. (This argument was for- merly called <i>force-p</i> . The old name is currently still accepted for compatibility.)

	simulate	If nil or not present then compile-system works silently. Otherwise a plan of the actions which compile-system intends to carry out is printed. What happens next depends on the value of <i>simulate</i> :
		t — do nothing.
		:ask — you are asked if you wish the plan to be carried out using y-or-n-p.
		<pre>:each — compile-system displays each action in the plan one at a time, and asks you whether you want to carry out this par- ticular action. The answer c executes the rest of the plan without further prompting, returns from compile-system without fur- ther processing, and y and n work as expected.</pre>
		:simulate may be abbreviated as :sim.
	load	If t then load-system is called after com- pile-system has finished. If :no then no files are loaded at all. The default is nil.
	args	Arguments to be passed directly to the com- piler.
	target-directory	This must be a string representing a valid directory. It defaults to the :default- pathname option to defsystem. This is the directory where the object files created are put. If the <i>target-directory</i> is given then dependency information expressed in the system rules is ignored. :target-directory may be abbreviated as :t-dir.
Values	compile-system returns nil.	
Examples	(compile-system 'blackboard :simulate :ask)	

(compile-system 'tms :load t)
(compile-system 'packages :load :no
 :target-directory "/usr/users/386i/")

Notes If the :load keyword is set to t then by default load-system is called after compile-system. This behavior can be changed to loading any file immediately after it is compiled by setting the variable defsystem::\*load-when-compile\* to non-nil.

See also defsystem load-system

## concatenate-system

Package	scm		
Syntax	concatenate-system output-file &rest args		
Arguments	In the following list of arguments, the words in parentheses are equivalent forms for the keyword:		
	output-file	The name of the required concatenated fasl.	
	simulate (sim)	Verbosity conditions, see Description for more detail.	
	force (force-p)	If $\mathbf{t}$ , then all files in the system will be concatenated.	
	source-only (source-p)		
		If $\mathbf{t}$ , the source files of the system are concatenated.	
	target-directory (t-dir)		
		The directory to search for the object files.	
	target-machine (t-machine)		

Description This function produces a single, concatenated fasl, *output-file*, from a list of individual systems (named amongst the *args*).

Since concatenated fasl files may be produced in this way, you do not need to be wary of PC filename conventions if developing sources on UNIX for a PC application. This clearly allows more freedom for naming source files. However, *output-file* must, in such cases, be a PC-compatible filename.

If *simulate* is nil or is not present, concatenate-system will work silently. Otherwise, a plan of the actions which concatenate-system intends to carry out is printed. What happens next depends upon the value of *simulate*:

- If it is t, the function does nothing.
- If :ask, then the user is asked, using y-or-n-p, if the plan should be carried out.
- If it is :each, the user is asked at each stage in the plan if the current action should be carried out. The responses y and n work as normal. If e is typed, concatenate-system exits without further processing.

If source-only is t, files will be loaded only if they are sources.

If, when searching *target-directory* for an object file, the file cannot be found, the appropriate source file from the system's default directory will be loaded instead.

The *target-machine* argument allows the concatenation of fasl files for loading on another machine. Possible values are:

- :harlequin-pc-lisp or :pc concatenate PC Lisp-Works fasl files.
- :harlequin-unix-lisp or :unix concatenate fasl files for the current UNIX machine.

By default, the concatenated fasl is built for the current machine architecture

defadvice		Macro	
Summary	Defines a new piece of advice.		
Package	lispworks		
Signature		pec name advice-type &key where documentation) mbda-list &body body => nil	
		me   -name   nod generic-fn-name [(class*)])	
	advice-type ::=	:before  :after  :around	
Arguments	dspec	Specifies the functional definition to which the piece of advice belongs. There are three forms which this specification may take. The first one above specifies a function by its name; the second one specifies a macro by name; the third specifies a method by the name of its generic function and by a list of classes to specialize the arguments to the method. In the case of a method the list of classes must correspond exactly to the classes of the specialized parameters of an existing method, and the advice is then attached to this method.	
		When advice is provided for a macro using defadvice, then the function with which the advice is associated is the expansion function for that macro. Thus before and after advice for a macro receive the arguments given to the macro expansion function, which are normally the macro call form and an environment.	

name	A symbol naming the piece of advice being created. It should of course be unique to the advised function, but does not need to be globally unique.
advice-type	A keyword specifying the kind of advice wanted.
where	Specifies where this advice should be placed in the ordering of pieces of advice for the function. By default a piece of advice is placed at the start of the corresponding sec- tion. If this argument is present and is :end then the advice is instead placed at the end of its section. The other permissible value for this argument is :start, which places the advice at the start of its section in the ordering (as in the default behavior).
documentation	A string providing documentation on the piece of advice.
lambda-list	A lambda list for the piece of advice. In the case of before and after advice this should be compatible with the lambda-list for the original definition, since such advice receives the same arguments as that function.
body	The main body of the advice.

Description defadvice is the macro used to define a new piece of advice. Advice provides a way to change the behavior of existing functional definitions in the system. In a simple instance advice might be used to carry out some additional actions before or after the original definition. More sophisticated uses allow the definition to be replaced by new code that can access the original function repeatedly or as rarely as desired, and that can receive different numbers of arguments and

Values

return any values. A function may have any number of pieces of advice attached to it by using defadvice.

There are three kinds of advice that may be defined: before, after and around advice. The first two kinds attach auxiliary code to be carried out alongside the original definition (before it for before advice, after it in the case of after advice). Around advice replaces the function altogether; it may define code that never accesses the original definition, that receives different numbers of arguments, and returns different values. All the pieces of advice for a function are ordered. The ordering is important in determining how all the pieces of advice for a function are combined. Around advice always comes first, then before advice, then the original definition, and lastly the after advice.

Conceptually the before advice, the original definition and the after advice are amalgamated into one new construct. If this gets called then each of its components receives the same arguments in turn, and the values returned are those produced by the last piece of after advice to be called in this way (or the original function if there is no after advice). The code associated with before and after advice should not destructively modify its arguments.

If around advice is present then the first piece of around advice is called, instead of the combination involving before and after advice discussed above. It does not have to access any of the other advice, nor the original definition. Its only link to the rest of the advice is by means of a call to callnext-advice. It may invoke this as often as it chooses, and by doing so it accesses the next piece of around advice if present, or else it accesses the combination of before and after advice together with the original definition.

Notes defadvice is an extension to Common Lisp.

## \*default-action-list-sort-time\*

Summary	Determines when actions in action lists are sorted.
Package	lispworks
Signature	*default-action-list-sort-time*
Default value	:execute
Description	Contains a keyword that is either :execute or :define- action, denoting when actions in action-lists are sorted (see define-action-list for an explanation of ordering specifi- ers). Actions are sorted either at time of definition (:define- action) or when their action-list is executed (:execute). The default sort time is :execute.

See also

## \*default-character-element-type\*

Parameter

Summary Provides defaults for all character type parameters.

Package lispworks

Description This variable provides defaults for all character type parameters. The legal values are base-char, lw:simple-char, and character. Its value must only be set via a call to lw:setdefault-character-element-type.

> This is intended mainly for running old 8-bit applications efficiently. If you write for a fat character implementation you should already be aware of these issues, and make some attempt to provide explicit types.

> The compiler always behaves as if this variable was bound to character; if you want assumptions about types to be hard-

#### The LISPWORKS Package

coded into your program, you must supply explicit declarations and type arguments.

See also string open set-default-character-element-type

## define-action

Macro

Summary	Adds a new action to a specified list.		
Package	lispworks		
Signature	<pre>define-action name-or-list action-name data &amp;rest specs =&gt;</pre>		
Arguments	name-or-list action-name data specs	A list or action list object. A general lisp object. An object. A list.	
Description	The define-action macro adds a new action to the specified list; this action will be executed according to the action-list's execution-function (see execute-actions) when executed. If the action-list specified by <i>name-or-list</i> does not exist, then this is handled according to the value of *handle-missing- action-list*.		
	<i>name-or-list</i> is evaluated to give either a list UID (to be looked up in the global registry of lists) or an action list object. <i>action-</i> <i>name</i> is a UID (general lisp object, to be compared by equalp). It uniquely identifies this action within its list (as opposed to among all lists).		
	1		

data specifies an object referring to data relevant to the action.

*specs* is a free-form list of ordering specifiers and extra keywords, used to control more details of how and when this action is executed.

Action-items are normally expected not to be redefined. If an action-item with that action-name already exists in the action-list (that is, one with an identifier equalp to the action-name), then the notification and subsequent handling of this attempt is controlled by the values in the list \*handle-exist-ing-action-in-action-list\*. This is to prevent problems due to re-evaluating an action definition inappropriately. Notification and redefine behavior can be overridden by using the :force keyword argument. In this case, any required redefinition is performed unconditionally and without notification.

The following keywords are recognized in the *specs* argument:

:after	The following element in <i>specs</i> is a UID. <b>:after</b> specifies that the action-item being defined must be run after the action-item named. If there is no action-item with a matching name, the restriction is ignored.
:before	Like <b>:after</b> , but this action-item must be run before the one specified.

**:after** and **:before** can be specified as many times as necessary to describe the ordering constraints of this action-item with respect to its neighbors.

:once	Specifies that this action-item should be exe- cuted only once; after execution, it is dis- abled.
:force	Specifies that this definition should override any previous definition of this action-item, rather than be subject to the value of *han- dle-existing-action-in-action-list*.

```
Example (define-action :network-startup "Reset decnet buffers"
'(decnet::reset-network-buffers
*net-buffers*)
:after "Reset core network"
:once))
```

See also undefine-action

## define-action-list

Macro

Summary	Defines a registered action list.		
Package	lispworks		
Signature	define-action-list uid &key documentation sort-time dummy- actions default-order execution-function =>		
Arguments	uid A general lisp object.		
	documentation	A string.	
	sort-time	One of :execute or :define-action.	
	dummy-actions	A list.	
	default-order	A list.	
	execution-functionA function.		

Description The define-action-list macro defines an action list.

*uid* is a unique identifier, and must be a general lisp object, to be compared by equalp. It names the list in the global registry of lists. See make-unregistered-action-list to create unnamed, "unregistered" action-lists. The *uid* may be quoted, but is not required to be. It is possible, but not recommended, to define an action-list with unique identifier nil. If a registered action-list with the *uid* already exists (that is, one which returns t when compared with equalp), then notification and subsequent handling is controlled by the value of the \*handle-existing-action-list\* variable. The *documentation* string allows you to provide documentation for the action list.

sort-time is a keyword specifying when added actions are sorted for the given list — either :execute or :defineaction (see \*default-action-list-sort-time\*).

*dummy-actions* is a list of action-names that specify placeholding actions; they cannot be executed and are constrained to the order specified in this list, for example

'(:beginning :middle :end)

*default-order* specifies default ordering constraints for subsequently defined action-items where no explicit ordering constraints are specified. An example is

```
'(:after beginning :before :end)
```

*execution-function* specifies a user-defined function accepting arguments of the form:

(the-action-list other-args-list &rest keyword-value-pairs)

where the two required arguments are the action-list and a list of additional arguments passed to execute-actions, respectively. The remaining arguments are any number of keyword-value pairs that may be specified in the call to execute-actions. If no execution function is specified, then the default execution function will be used to execute the action-list.

See also \*default-action-list-sort-time\* \*handle-existing-action-list\* undefine-action-list

## defsystem

Macro

Summary defsystem is used to define systems for use with the Lisp-Works system tools. A system is a collection of files and other systems that, together with rules expressing the interdependencies of those files and subsystems, make a complete program. The LispWorks system tools support the development and maintenance of large programs.

Package	lispworks		
Signature	defsystem system-name options &key members rules => system		
Arguments	system-name	The name of the system to be made.	
	<i>options</i> are expressed as a list of keyword argument pairs. The following keywords are recognized:		
	:package	The default package that files are compiled and loaded into. If not specified this defaults to the USER package.	
	:default-pathname		
		The default pathname in which to find files. defsystem checks that all the files given as members actually exist.	
	:default-host	The root pathname of a system is defined to be the :default-host if it is given. Other- wise, it is taken to be the directory contain- ing the defsystem file.	
		Absolute pathnames are interpreted liter- ally, and relative pathnames are taken rela- tive to the root pathname.	
		Note that you no longer need to use *load- truename* as the default pathname.	
	:default-type	This is the default type of the members of the system. This may be :lisp-file, :c- file, or :system. The default is :lisp- file.	

:documentation This is a string.

*members* is a list defining the members of the system. Each element of the list may be a symbol or a string representing the name of the physical file referred to, or a list of format (*name {keyword value}\**) where *name* is once again a symbol or a string referring to the physical file, and the possible keywords are:

:type	The type of this member. If not specified it defaults to the value of :default-type given as an <i>option</i>
:root-module	If nil then this member is not loaded unless its loading is specifically requested as a result of a dependency on another module
:source-only	Only the source file for this member is ever loaded
:load-only	The member is never compiled by defsys- tem, objects are loaded in preference to source files

## :load-for-compile-only

The member is only loaded as necessary during compilation and is never loaded independently

:features The member is only considered during planning if the feature expression is true

rules is a list of rules of the following format :

```
(:in-order-to action {:all | ({ member-name }* })
  (:caused-by {(action {:previous |{member-name }* }) }*)
  (:requires {(action {:previous |{ member-name }*}) }*))
```

The keyword :all refers to all the members of the system. It provides a shorthand for specifying that a rule should apply to all the system's members. The keyword :previous refers to all the members of the system that are before the member in the list of members. This makes it easy, for example, to

```
specify that in order to compile a file in a system, all the
              members that come before it must be loaded.
Values
             The name of the system is returned.
Examples
              (hcl:defsystem defsys-macros
                 (:default-pathname "/usr/users/james/scm/defsys/"
                  :default-type :lisp-file
                  :package defsystem)
                 :members ("new-macros"
                           "scm-timemacros"))
              (hcl:defsystem clos-sys
                 (:default-pathname "/usr/users/clc/defsys/"
                  :default-type :lisp-file
                  :package defsystem)
                 :members
                   (("defsys-macros" :type :system :root-module nil)
                     "class"
                     "time-methods"
                    ("scm-pathname" :source-only t)
                     "execute-plan"
                     "file-types"
                     "make-system"
                     "conv-defsys")
                 :rules
                   ((:in-order-to :compile ("class" "time-methods")
                         (:caused-by (:compile "defsys-macros"))
                                      (:requires
                                        (:load "defsys-macros")))
                    (:in-order-to :compile
                                      ("time-methods" "execute-plan")
                                      (:requires (:load "class")))))
```

	(hcl:defsystem dataworks-demo
	(:default-pathname
	(directory-namestring system::*load-pathname*)
	:default-type :system)
	:members (
	"db-class"
	"planar"
	"dataworks-dep"
	"dataworks-interface-tk"
	"dataworks-interface-tools"
	"drugs-demo"
	("gen-demo" :type :lisp-file)
	("load-icon" :type :lisp-file :source-only t)
	)
	:rules ((:in-order-to :compile :all
	(:requires (:load :previous)))))
Notes	Systems that are members of another system must be
	declared in the system declaration file before the system of
	which they are a part.
	which they are a part.
	The ordering of members is important and reflects the order
	in which operations are carried out on the members of the
	-
	system.
See also	load-system
	compile-system

# dotted-list-length

Summary	Similar to list-	length
Package	lispworks	
Signature	dotted-list-le	ngth list => result
Arguments	list	A list.
Value	result	An integer.

## The LISPWORKS Package

The function dotted-list-length performs the same action
as list-length, except that if the last cdr is not nil then
instead of signalling an error, it returns the number of conses
plus 1.

See also dotted-list-p

# dotted-list-p

## Function

Summary	Tests whether a	cons is a list ending in a non-nil cdr.
Package	lispworks	
Signature	<pre>dotted-list-p list =&gt; bool</pre>	
Arguments	list	A list, which must be a cons.
Values	bool	A boolean.
Description	The function dotted-list-p is a predicate which tests whether <i>list</i> (which must be a cons) is a list ending in a non- nil cdr. It returns t if this is the case, otherwise it returns nil.	
See also	dotted-list-lo	ength

# do-nothing

Summary	Ignores its argui	ments and returns an unspecified value.
Package	lispworks	
Signature	do-nothing &rea	st ignore => unspecified
Arguments	ignore	All arguments are ignored.

Values	<i>unspecified</i> An unspecified value.
Description	The function do-nothing ignores its arguments and returns an unspecified value. It is useful as a function argument.
See also	false true

## environment-variable

Summary	Reads the value of an environment variable from the envi- ronment table of the calling process.		
Package	lispworks		
Signature	environment-variable name => result		
Arguments	name	A string.	
Values	result	A string or nil.	
Description	The function environment-variable reads the environment variable specified by <i>name</i> and returns its value, or nil if the variable could not be found. A setf method is also defined, allowing you to set the value of an environment variable: (setf (environment-variable <i>name</i> ) value)		
	If <i>value</i> is a string, then <i>name</i> is set to be <i>value</i> . If <i>value</i> is nil then <i>name</i> is removed from the environment table.		
Example	In this first exam PATH is returned	nple the value of the environment variable l:	
	(environment-v	ariable "PATH")	
	The result is a string of all the defined paths:		

```
"c:\\hqbin\\nt\\x86;c:\\hqbin\\nt\\x86\perl;c:\\hqbin\
\win32;c:\\usr\\local\\bin;C:\\WINNT35\\system32;C:\\WI
NNT35;;C:\\MSTOOLS\\bin;C:\\TGS3D\\PROGRAM;c:\\program
files\\devstudio\\sharedide\\bin\\ide;c:\\program
files\\devstudio\\sharedide\\bin;c:\\program
files\\devstudio\\vc\\bin;c:\\msdev\\bin;C:\\WINDOWS;C:
\\WINDOWS\\COMMAND;C:\\WIN95\\COMMAND;C:\\MSINPUT\\MOUS
E"
```

In the second example, the variable MYTZONE is found not to be in the environment table:

```
(environment-variable "MYTZONE")
```

NIL

It is set to be GMT using the setf method:

(setf (environment-variable "MYTZONE") "GMT")

## execute-actions

Macro

Summary	Executes in sequence the actions on a given list.		
Package	lispworks		
Signature	<pre>execute-action other-args =&gt;</pre>	s name-or-list &rest keyword-value-pairs &rest	
Arguments	name-or-list	An action list	
	keyword-value-pairs		
	See description.		
	other-args	A list.	
Description	The execute-actions macro executes, in sequence, the actions on the specified list. If the action-list specified by <i>name-or-list</i> does not exist, then this is handled according to		

the value of \*handle-missing-action-list\*. Note that *name-or-list* is evaluated.

If a user-defined execution function was specified when the action list was defined, then it should accept the following arguments:

## (action-list other-args &rest keyword-value-pairs)

Note that *other-args* is passed as a single list.

If a user-defined execution function was not specified when the action list was defined, then the following default mapping occurs. Each action's data is invoked via apply on other-args:

### (apply data other-args)

This behavior is modified by the keyword-value-pairs, thus:

- If the keyword parameter :ignore-errors-p is non-nil, any otherwise-unhandled errors signalled inside the execution of that function will be trapped, and a warning issued. Execution continues with the next action-item. If :ignore-errors-p is nil (or not specified), then the error is not trapped.
- If the keyword parameter :post-process is non-nil, the first value returned by each action is handled, according to :post-process, thus:

:collect	collect values into list
:and	return t only if all values are t. Return nil immediately if any value is nil
:or	return first non-nil value

## See also define-action with-action-list-mapping

## extended-char

Summary	The extended character type.
Package	lispworks
Signature	extended-char
Description	The type of extended characters. A synonym for extended- character, but with standard spelling.

## extended-character

Summary	The extended character type.
Package	lispworks
Signature	extended-character
Description	The type of extended characters.

# extended-character-p

## Function

Туре

Туре

Summary	Tests if an object is an extended character.		
Package	lispworks		
Signature	<pre>extended-character-p object =&gt; bool</pre>		
Arguments	object	The object to be tested.	
Values	bool	t if <i>object</i> is an extended character; nil otherwise.	
Description	This is the predicate for extended characters.		

See also extended-character

## extended-char-p

## Function

Summary	Tests if an object is an extended character.		
Package	lispworks		
Signature	<pre>extended-char-p object =&gt; bool</pre>		
Arguments	object	The object to be tested.	
Values	bool	t if <i>object</i> is an extended character; nil otherwise.	
Description	This is also the predicate for extended characters, only with standard spelling.		
See also	extended-char extended-chara	cter-p	

## \*external-formats\*

Variable

- Summary A list of the names of the defined external formats.
- Package lispworks
- Initial Value (win32:code-page fli::latin-1-wchar fli:ascii-wchar external-format:double-byte-table-lookup :euc-jp :sjis :latin-1-terminal :unicode :latin-1-checked :euc :shift-jis :nihongo-ms :nihongo-euc :nihongo-jis character external-format::raw-simple-character external-format::raw-base-character :ascii-terminal :ascii :latin-1)

false		Function
Summary	Ignores its arguments and returns nil.	
Package	lispworks	
Signature	false &rest <i>ignore</i> -> nil	
Arguments	<i>ignore</i> All arguments are ignored.	
Value	nil	
Description	The function false takes any number of arguments ignores, and returns mil. It is useful as a functional ment.	
See also	do-nothing true	

# file-directory-p

Summary	Tests for the presence of a directory.		
Package	lispworks		
Signature	<pre>file-directory-p pathname =&gt; bool</pre>		
Arguments	pathname	A pathname, string, or file-stream.	
Values	bool	If t, the pathname represented by pathname exists and is a directory. If nil, it either does not exist, or it is not a directory.	
Description	file-directory directory.	r-p tests whether the pathname represents a	

Examples	CL-USER T	70	>	(file-directory-p	"~")
	CL-USER NIL	71	>	(file-directory-p	".login")

## function-lambda-list

## Function

Summary	Returns the	Returns the arguments of the given function.	
Package	lispworks		
Signature	function-la	mbda-list function & optional error-p => args	
Arguments	function	The name of the function whose arguments are required.	
	error-p	If <i>error-p</i> is nil, the function returns :none if the function is not defined, and does not start the debugger.	
Values	args	The arguments of the given function.	
Example	command)	TEST 2 > (function-lambda-list 'editor:create-buffer- command) (EDITOR::P &OPTIONAL EDITOR:BUFFER-NAME)	

# get-gc-parameters

Summary	Returns the current values of various garbage collector parameters.
Package	lispworks
Signature	get-gc-parameters &key parameters => values

Arguments	parameters	If :full, full information for all GC parame- ters is returned. Otherwise, this should rep- resent a single GC parameter.	
Values	values	If parameters is :full, this is an associated list containing every GC parameter, together with its current value. If parameters speci- fies a GC parameter, only the value of that parameter is returned.	
Description	See set-gc-parameters for a full description of these parameters. With keyword argument, of one of the parameters, the corresponding value is returned.		
Example	CL-USER 44 : 1 > (get-gc-parameters :minimum-overflow) 500000 CL-USER 45 : 1 > (get-gc-parameters :full) ((:ENLARGE-BY-SEGMENTS . 10) (:MINIMUM-FOR-PROMOTE . 1000) (:MAXIMUM-OVERFLOW . 1000000) (:MINIMUM-OVERFLOW . 500000) (:MINIMUM-BUFFER-SIZE . 200) (:NEW- GENERATION-SIZE . 4096) (:PROMOTE-MAX-BUFFER . 100000) (:PROMOTE-MIN-BUFFER . 200) (:MAXIMUM-BUFFER-SIZE . 131072) (:MINIMUM-FOR-SWEEP . 8000) (:BIG-OBJECT . 131072) (:SMALL-OBJECT . 100))		
See also	set-gc-parameters		

## get-inspector-values

**Generic Function** 

Summary Customizes the information displayed in the Common LispWorks inspector tool.

Package lispworks

Signature get-inspector-values object mode

Arguments	object	The object to be inspected.
	mode	Name of a mode, or nil. nil defines the default inspection format for object.
Values	and <i>values</i> are the inspector winde	ues: <i>names</i> , <i>values</i> , <i>getter</i> , <i>setter</i> and <i>type</i> . <i>names</i> he two lists displayed in columns in the ow. <i>getter</i> is ignored. <i>setter</i> is a function used values. <i>type</i> is displayed at the foot of the ow.
Description	This generic function allows you to customize the Common LispWorks Inspector by adding new formats (corresponding to different values of mode) in which instances of a particular class can be inspected. Mode nil is the default mode, which is always present (it can be overwritten).	
	LispWorks inclu	udes methods for:
	(get-inspector (get-inspector (get-inspector	-values (object nil)) -values (standard-object nil)) -values (structured-object nil)) -values (sequence nil)) -values cons nil))
	and so on.	
Example	only direct slots	llows inspection of a CLOS object, displaying s form a chosen class in its class precedence useful when an object inherits many slots

from superclasses, and the inherited slots are of no interest.

```
(defmethod lispworks:get-inspector-values
         ((object standard-object)
          (mode (eql 'direct-as)))
         (declare (ignore mode))
         (loop with object-class =
                (class-of object)
               with precedence-list =
                (class-precedence-list object-class)
               with items =
                (loop for super in precedence-list
                      collecting (list*
                                  (format nil "~a"
                                   (class-name super))
                                   super))
               with class =
                (or (capi:prompt-with-list items
                        "Direct slots as ...")
                    object-class)
                            ;; default if no selection
               with slots =
                (class-direct-slots class)
               for slot in slots
               for name =
                    (clos::slot-definition-name slot)
               collect name into names
               collect (if (slot-boundp object name)
                           (slot-value object name)
                         :slot-unbound)
               into values
               finally
               (return
                (values
                 names
                 values
                 nil
                 #'(lambda
                    (x slot-name index new-value)
                    (declare (ignore index))
                    (setf (slot-value x slot-name)
                          new-value))
                 (format nil "~a - direct slots as ~a"
                         (class-name object-class)
                         (class-name class))))))
```

# \*handle-existing-action-in-action-list\*

Variable

Summary	Contains keywords determining behavior on exceptions raised when an action definition already exists in a given action list.
Package	lispworks
Signature	*handle-existing-action-in-action-list*
Default value	'(:warn :skip)
Description	A list containing one of :warn, or :silent, determining whether to notify the user, and one of :skip, or :redefine, to determine what to do about an action definition when the action already exists in the given action-list. The default value is '(:warn :skip). It is used by define-action.
See also	define-action-list

# \*handle-existing-action-list\*

Variable

Summary	Contains keywords determining what to do about a given action list operation when the action list already exists.
Package	lispworks
Signature	*handle-existing-action-list*
Default value	'(:warn :skip)
Description	A list containing either :warn or :silent, determining whether to notify the user, and either :skip or :redefine to determine what to do about an action-list operation when the action-list already exists. The default value is '(:warn :skip). It is used by the define-action-list macro.

See also define-action-list

## \*handle-missing-action-list\*

## Variable

- Summary Defines how to handle an operation on a missing action list.
- Package lispworks
- Signature \*handle-missing-action-list\*
- Default Value :error
- Description A keyword; one of :warn, :error, or :ignore, denoting how to handle an operation on a missing action-list. The default value is :error. It is used by undefine-action-list, printactions, execute-actions, define-action and undefineaction.
- See also define-action-list

## \*handle-missing-action-in-action-list\* Variable

- Summary Denotes how to handle an operation on a missing action.
- Package lispworks
- Initial value :warn
- Description A keyword; one of :warn, :error or :ignore, denoting how to handle an operation on a missing action. Its default value is :warn. It is used by undefine-action.
- See also define-action-list

## \*handle-warn-on-redefinition\*

# SummarySpecifies the action on redefining a symbol.PackagelispworksInitial value:warnDescription\*handle-warn-on-redefinition\* specifies what action<br/>should be taken on finding redefined external symbols in the<br/>packages specified in the variable \*packages-for-warn-on-<br/>definition\*. If \*handle-warn-on-redefinition\* is set to<br/>:warn then you are merely warned. If, however, it is set to<br/>:error, then LispWorks signals an error.See also\*packages-for-warn-on-redefinition\*

## hardcopy-system

## Function

Variable

Summary	Print each file of a system to a printer.	
Package	lispworks	
Signature	hardcopy-system system-name &key command simulate => nil	
Arguments	system-name	A symbol representing the name of the sys- tem. The system must have been defined using the defsystem macro.
	simulate	If nil or not present then hardcopy-system works silently. Otherwise a plan of the actions which hardcopy-system intends to carry out is printed. What happens next depends on the value of <i>simulate</i> :
		t — do nothing.

**:ask** — you are asked, using **y-or-n-p**, if you want the plan to be carried out.

:each — hardcopy-system displays each action in the plan one at a time, and asks you if you want to carry out this particular action. The answer executes the rest of the plan without further prompting, e returns from hardcopy-system without further processing, and y and n work as expected.

Values	hardcopy-system returns nil.
Examples	(hardcopy-system 'blackboard) (hardcopy-system 'tms :simulate :ask :command "lpr")
Notes	By default, hardcopy-system uses *print-command* as the command sent to the shell.
See also	defsystem compile-system

## \*lispworks-directory\*

load-system
\*print-command\*

## Variable

Summary	The main LispWorks installation directory.
Package	lispworks
Initial value	Usually C:\Program Files\Harlequin\LispWorks\current- lib though this is determined by the entry in the configura- tion file.
Description	Holds the name of the directory where various files impor- tant for the running of LispWorks are located.

The subdirectories of \*lispworks-directory\* should include: config, which contains the configuration files. patches, which contains any patches that are sent by The Harlequin Group Limited. postscript, which contains configuration files for printing using the CAPI printing library. See the chapter on Simple Customization in the *LispWorks User Guide* for more information on printer configuration. examples, which contains various files of example code. Other directories are app-defaults, etc, gnu-lisp, ilisp, load-on-demand, manual, old-environment, and tk-resources.

## load-all-patches

Summary	Loads all patch files into the image.
Package	lispworks
Signature	load-all-patches => nil
Arguments	None.
Values	Returns nil.
Description	Loads into the image all appropriate files from the directory patches in the directory determined by *lispworks-direc-tory*.When the appropriate patches have successfully been loaded, the updated version of the image can be saved using save-image.

Normally, you call load-all-patches before starting the Common LispWorks environment. Thus, you normally place the call to this function in your .lispworks file.

The system expects all patches to be loaded sequentially. If a patch is missing, there is a warning message. In this situation, it is advisable to contact Harlequin to obtain a copy of the missing patch.

## load-system

Summary	Load each file of a system into the Lisp image if either the file has not been loaded, or the file has been written since it was last loaded.	
Package	lispworks	
Signature	<pre>load-system system-name &amp;key force simulate source-only target- directory =&gt; nil</pre>	
Arguments	system-name	A symbol representing the name of the sys- tem. The system must have been defined using the defsystem macro.
	force	If t then all the files in the system are loaded regardless. (This argument was formerly called <i>force-p</i> . The old name is currently still accepted for compatibility.)
	simulate	If nil or not present then load-system works silently. Otherwise a plan of the actions which load-system intends to carry out is printed. What happens next depends on the value of <i>simulate</i> :
		t — do nothing.
		:ask — you are asked, using y-or-n-p, if you want to carry out the plan.

		<pre>:each — load-system displays each action in the plan one at a time, and asks you if you want to carry out this particular action. The answer executes the rest of the plan without further prompting, e returns from load- system without further processing, and y and n work as expected.</pre>
	source-only	If t the source files of the system are loaded. This only applies to file types where it makes sense to load a source file.
	target-directory	This is the directory to search for the object files. If the object file cannot be found here then the source file from the system's default directory are loaded.
Values	load-system re	turns nil.
Examples	(load-system 'blackboard)	
	(load-system '	<pre>tms :simulate :ask :source-only t)</pre>
Notes	For Lisp files load-system loads the object file (if it exists) into the image, unless over-ridden by the :source-only key- word argument. This behavior can be changed so that the newest file (whether source or object) is loaded by setting the variable *load-source-if-newer* to t.	
See also	defsystem compile-syste	m

# make-unregistered-action-list

Function

Summary Makes an unregistered action list.

Package lispworks

## The LISPWORKS Package

Signature	<pre>make-unregistered-action-list &amp;key documentation sort-time dummy-actions default-order execution-function =&gt;</pre>		
Arguments	documentation	A string.	
	sort-time	One of :execute or :define-action.	
	dummy-actions	A list.	
	default-order	A list.	
	execution-functio	<i>n</i> A function.	
Description		n-list not registered in the global registry of ord arguments are as for define-action-list.	
	The <i>documentation</i> string allows you to provide documenta- tion for the action list.		
	<i>sort-time</i> is a keyword specifying when added actions are sorted for the given list — either :execute or :define-action (see *default-action-list-sort-time*).		
	<i>dummy-actions</i> is a list of action-names that specify place ing actions; they cannot be executed and are constraine the order specified in this list, for example		
	'(:beginning :	niddle :end)	
	<i>default-order</i> specifies default ordering constraints for subse- quently defined action-items where no explicit ordering con- straints are specified. An example is		
	'(:after beginning :before :end)		
	<i>execution-function</i> specifies a user-defined function accepting arguments of the form:		
	(the-action-list other-args-list &rest keyword-value-pairs)		
	where the two required arguments are the action-list and a list of additional arguments passed to execute-actions, respectively. The remaining arguments are any number of keyword-value pairs that may be specified in the call to execute-actions. If no execution function is specified, then		

the default execution function will be used to execute the action-list.

See also	define-action-list
	*handle-warn-on-redefinition*

## print-actions

## Function

Summary	Generates a listing of the action items on a given action list in order.	
Package	lispworks	
Signature	print-actions name-or-list &optional stream	
Arguments	name-or-listAn action list.streamA stream.	
Description	Generates a listing of the action items on this action-list, in order. If the action-list specified by <i>name-or-list</i> does not exist, then this is handled according to the value of *handle- missing-action-list*. <i>stream</i> is an optional argument spec- ifying where to print the output.	
See also	print-action-lists	

## print-action-lists

- Summary Prints a list of all the actions lists in the global registry.
- Package lispworks
- Signature print-action-lists & optional stream
- Arguments stream A stream.

Generates a listing of all the action lists in the global registry.
The ordering of the action lists is random. <i>stream</i> is an
optional argument specifying where to print the output.

See also print-actions

# \*print-command\*

## Variable

Summary	Used to hold the command sent to Lispworks for all printing operations.
Package	lispworks
Initial Value	Initially, this variable is unbound.
Description	This variable is used as the command sent by LispWorks to the shell in all printing operations. For example, it is used by the Editor command M-x Print Buffer and lw:hardcopy-system.

## rebinding

## Macro

- Summary Ensures unique names for all the variables in a groups of forms.
- Package lispworks

Signature	rebinding (&re	st vars) &body body => form
Arguments	vars	The variables to be rebound.
	body	A body of forms, the variables in which

should be unique.

Values	Returns the body wrapped in a form that creates unique names for each variable.
Description	Returns the <i>body</i> wrapped in a form which creates a unique name for each of the variables (compare with gensym) and binds these names to the values of the variables. This ensures that the body can refer to the variables without name clashes with other variables elsewhere.
Example	After defining
	<pre>(defmacro lister (x y)   (rebinding (x y)         `(list ,x ,y)))</pre>
	the form (lister i j) macroexpands to
	(LET* ((#:X-77 I) (#:Y-78 J)) (LIST #:X-77 #:Y-78))
See also	with-unique-names

remove-advice	

Summary	Remove a piece	e of advice.
Package	lispworks	
Signature	dspec ::= fn-na macr	dspec name => nil nme   o-name   chod generic-fn-name [(class*)])
Arguments	dspec	Specifies the functional definition to which the piece of advice belongs. The specifica- tion contains the name of the associated function. In the case of a method the list of classes is used to identify from which partic-

		ular method the advice should come. This list must correspond exactly with the classes corresponding to the specialized parameters for some method belonging to the generic function.
	name	A symbol naming the piece of advice to be removed. Since several pieces of advice may be attached to a single functional definition, the name is necessary to indicate which one is to be removed.
Values	remove-advice	returns nil.
Description	advice. Advice Pieces of advice vice. They defit the function is instead of the f original definit functions, advi (in this case it is function). hcl:delete-ad	is the function used to remove a piece of is a way of altering the behavior of functions. e are associated with a function using defad- ine additional actions to be performed when invoked, or alternative code to be performed unction, which may or may not access the ion. As well as being attached to ordinary ce may be attached to methods and to macros in fact associated with the macro's expansion wice is a macro, identical in effect to remove- that you do not need to quote the arguments.
Notes	remove-advice	is an extension to Common Lisp.
See also	delete-advice	1
removef		Macro
Summary	Removes an ite	em from a sequence.
Package	lispworks	

Signature	removef place i	tem &key test test-not start end key => result
Arguments	place	A place.
	item	An object.
	test	A test function.
	test-not	A test function.
	start	An integer.
	end	An integer or nil.
	key	A key function.
Values	result	A sequence.
Description		macro removef removes an item from a gremove. See remove for more details.
See also	appendf	
sbchar		Function
Summary	The accessor fo	r simple base strings.
Package	lispworks	
Signature	sbchar simple-	-base-string index => value
Arguments	index	An index.
Values	value	The new value of the simple base string.
Description	This is the acce	ssor for simple base strings. setf is allowed.
See also	simple-base-s	tring

# set-default-character-element-type

Function

Summary	Sets the value	Sets the value of lw:*default-character-element-type*.		
Package	lispworks			
Signature	set-default-c	<pre>set-default-character-element-type type =&gt; type-defaults</pre>		
Arguments	type	A character type. This can take any of the values base-char; lw:simple-char and character		
Values	type-defaults	The new value of lw:*default-character- element-type*.		
Description	Sets the value of <pre>lw:*default-character-element-type*, ensuring that the system's internal state is also updated accordingly.</pre>			
	If you are running an existing 8-bit application you will only need to have this in your site or user configuration file:			
	(lw:set-default-character-element-type 'base-char)			
	It would be a mistake to call this in a loadable package.			
	Hence we cons a parameter.	Hence we consider lw:*default-character-element-type* a parameter.		
See also	string open *default-cha:	racter-element-type*		

# simple-base-string-p

Function

ıg.
ļ

Package lispworks

Signature	<pre>simple-base-string-p object =&gt; bool</pre>	
Arguments	object	The object to be tested.
Values	bool	ד if <i>object</i> is a simple base string; mil other- wise.
Description	This is the predicate for simple base strings.	
See also	simple-base-string	

# simple-char

simple-char	$T_{j}$	ype
Summary	The simple character type.	
Package	lispworks	
Signature	simple-char	
Description	The type of simple characters (standard term for chars wit null implementation-defined attributes, that is, no bits).	h

# simple-char-p

Summary	Tests if an object is a simple character.	
Package	lispworks	
Signature	<pre>simple-char-p object =&gt; bool</pre>	
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is a simple character; nil other- wise.

## The LISPWORKS Package

- Description The predicate for simple characters.
- See also simple-char

## simple-text-string

# SummaryThe simple text string type.PackagelispworksSignaturesimple-text-string lengthArgumentslengthThe length of the string (or \*, meaning any).DescriptionThis is the simple version of text-string, that is, the string<br/>itself is simple. Equivalent to:<br/>(simple-vector lw:simple-char length)See alsotext-string

## simple-text-string-p

## Function

Type

Summary	Tests if an object is a simple text string.	
Package	lispworks	
Signature	<pre>simple-text-string-p object =&gt; bool</pre>	
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is a simple text string; nil other- wise.
Description	This is the predicate for simple text strings.	

stchar			Function
Summary	The accessor for simple text strings.		
Package	lispworks		
Signature	<pre>stchar simple-text-string index =&gt; value</pre>		
Arguments	index	An index.	
Values	value	The new value of the simple text s	string.
Description	This is the accessor for simple text strings. setf is allowed.		
See also	simple-text-string		

# string-append

Summary	Constructs a sin	gle string from a number of strings.
Package	lispworks	
Signature	string-append &rest strings => string	
Arguments	strings	Any number of strings.
Values	string	A string.
Description	The string-append function takes any number of strings and constructs a single string from them. Each of the elements of the <i>strings</i> argument are first coerced into a string using the string function if they are not already a string.	

text-string	Тура	e
Summary	The text string type.	
Package	lispworks	
Signature	text-string length	
Arguments	<i>length</i> The length of the string (or *, meaning any).	,
Description	The type of strings that can hold any simple character, that is, (vector lw:simple-char length). This is the string type that is guaranteed to always hold any character used in writ- ing text (program text or natural language), but not necessar- ily key event descriptors and other such character objects.	
	In the current HCL implementation, it is equivalent to 8-bit- string; in DBCS images, it is equivalent to 16-bit-string.	-
See also	8-bit-string 16-bit-string	

# text-string-p

Summary	Tests if an object is a text string.	
Package	lispworks	
Signature	<pre>text-string-p object =&gt; bool</pre>	
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is a text string; היו otherwise.
Description	This is the predicate for text strings.	
See also	text-string	

#### top-level-form

#### Macro

Summary	Use a specified name as a parent name in the specified body of forms.		
Package	lispworks		
Signature	top-level-form name &body body => result		
Arguments	name	The name to use as the parent.	
	body	The forms across which to use name as the parent.	
Values	value	The result of evaluating <i>body</i> .	
Description	This macro is useful in the expansion of defining macros. Every form in the <i>body</i> is given <i>name</i> as its parent name. This can be useful for things such as debugging.		
	top-level-form is exported from the LISPWORKS package.		
Example	Compiling the f	ile	
	(in-package "CL-USER")		
	<pre>(defmacro define-thing (name value)   `(top-level-form (define-thing ,name)       (install-thing ',name ,value)))</pre>		
	<pre>(defun install-thing (x y)   (set x y))</pre>		
	(define-thing the-thing 42) (define-thing some-thing 43)		
	gives messages	such as	
	; (DEFINE-THING THE-THING) ; (DEFINE-THING SOME-THING) rather than the more cryptic		

; (TOP-LEVEL-FORM 3)

which is the default.

In addition, LispWorks is able to find the source of this form if after typing Meta-. you enter

```
(define-thing the-thing)
```

#### true

#### Function

Summary	Ignores its arguments and returns t.		
Package	lispworks		
Signature	true &rest ignore => t		
Arguments	<i>ignore</i> All arguments are ignored.		
Values	t		
Description	The function $true$ ignores all its arguments and returns $t$ . It is useful as a functional argument.		
See also	do-nothing false		
See also	define-definition-spec		

#### undefine-action

#### Macro

Summary	Removes an action from a specified list.	
Package	lispworks	
Signature	undefine-actio	n name-or-list action-name =>
Arguments	name-or-list	A list or action list object.

	action-name A general lisp object.		
Description	The undefine-action macro removes the specified action from the specified list. If the action specified by action-name does not exist, then this is handled according to the value of *handle-missing-action-in-action-list*.		
	<i>name-or-list</i> is evaluated to give either a list UID (to be looked up in the global registry of lists) or an action list object. <i>action- name</i> is a UID (general lisp object, to be compared by equalp). It uniquely identifies this action within its list (as opposed to among all lists).		
See also	define-action		

#### undefine-action-list

Summary	Removes a given defined action list.
Package	lispworks
Signature	undefine-action-list uid =>
Arguments	<i>uid</i> A lisp object.
Values	None.
Description	The undefine-action-list flushes the specified list (and all its action-items). If the action-list specified by <i>uid</i> does not exist, then handling is controlled by the value of the *han-dle-missing-action-list* variable.
See also	define-action-list

Macro

when-let		Macro	
Summary	Executes a body of code if a form evaluates to non-nil, prop- agating the result of the form through the body of code.		
Package	lispworks	lispworks	
Signature	when-let (var form) & body body => result		
Arguments	var	A variable whose value is used in the evalu- ation of <i>body</i> .	
	form	A form, which must evaluate to non-nil.	
	body	A body of code to be evaluated condition- ally on the result of <i>form</i> .	
Values	result	The result of evaluating <i>body</i> using the value <i>var</i> .	
Description	This macro executes the body of code if the form evaluates to a non-nil result. Within the body, the variable <i>var</i> is bound to the result of the <i>form</i> .		
Example	The form		
	<pre>(when-let (position (search string1 string2))   (print position)) macroexpands to (let ((position (search string1 string2)))   (when position        (print position)))</pre>		

#### whitespace-char-p

Function

Summary	Tests is a character represents white space.

Package lispworks

Signature	whitespace-char-p char => bool	
Arguments	char	A character.
Values	bool	t if char represents white space; היו other- wise.
Description	<pre>This predicate recognizes [whitespace1], as described in the standard:</pre>	
See also	*extended-spaces*	

#### with-action-item-error-handling

#### Macro

Summary	Executes a body of code across action lists and items, signal- ling errors and then continuing to the next action item.	
Package	lispworks	
Signature	with-action-item-error-handling action-list-var action-item- var ignore-errors-p &body body	
Arguments	action-list-var	A variable.
	action-item-var	A variable.
	ignore-errors-p	A boolean.
	body	A body of Lisp code.
Description	The with-action-item-error-handling macro executes the <i>body</i> with <i>action-list-var</i> and <i>action-item-var</i> are bound to the action list and item respectively. If <i>ignore-errors-p</i> is set to t	

then errors are handled. The behavior of the handler is to signal a warning in which the action-list, item and original error are all reported; execution then continues with the next action-item.

```
Example (defun my-execution-function (the-action-list
other-args
&key ignore-errors-p
&allow-other-keys)
(with-action-list-mapping (the-action-list
an-action-item
action-item_data)
(with-action-item-error-handling (the-action-list
an-action-item
ignore-errors-p)
(do-something-interesting-first)
(apply (car action-item-data) other-args (cdr
action-item-data)))))
```

If this function was invoked with the keyword argument <code>:ignore-errors-p t</code>, and an error was signalled while executing the body-form(s) for one of the action-items, then a warning such as:

Warning: Got an error 'The variable \*PREV-STATE\* is unbound.' while executing action "Initialize State" in list "Startup Inits".

would be signalled and execution would continue with the next action-item.

See also \*handle-missing-action-in-action-list\*

#### with-action-list-mapping

Macro

Summary Maps over an action list's actions with given variables bound to the executing action and its data.

Package lispworks

Signature	with-action-list-mapping action-list item-var data-var &optional post-process &body body)		
Arguments	action-list An action list.		
	item-var	A Lisp symbol.	
	data-var	A Lisp symbol.	
	post-process	A keyword.	
	body	A body of Lisp code.	
Description	The with-action-list-mapping macro maps over an action- list's action-items. During execution, the symbols specified for <i>item-var</i> and <i>data-var</i> are bound to the executing action- item and its data respectively. See execute-actions for more on post-processing.		
	If this function is invoked with the keyword argument :post-process :collect, a list the values returned by each action-item's setf operation are returned.		
Examples	<pre>(defun my-execution-function (the-action-list other-args &amp;key (post-process nil) &amp;allow-other-keys) (declare (ignore other-args)) (with-action-list-mapping (the-action-list</pre>		
See also	execute-actio	ns	

#### with-unique-names

Summary	Returns a body similar name.	of code with each specified name bound to a
Package	lispworks	
Signature	with-unique-names (&rest <i>names</i> ) &body <i>body</i> => <i>result</i>	
Arguments	names	The names to be rebound in <i>body</i> .
	body	The body of code within which <i>names</i> are rebound.
Values	result	The result of evaluating <i>body</i> .
Description	Returns the body with each <i>name</i> bound to a symbol of a similar name (compare gensym).	
Example	After defining	
	<pre>(defmacro lister (p q) (with-unique-names (x y) `(let ((,x (x-function)) (,y (y-function))) (list ,p ,q ,x ,y))))</pre>	
	the form (lister i j) macroexpands to	
	(LET* ((#:X-88 (X-FUNCTION)) (#:Y-89 (Y-FUNCTION))) (LIST i j #:X-88 #:Y-89))	

See also rebinding

### 12

## The MP Package

This chapter describes symbols available in the MP package, giving you access to the multi-processing capabilities of LispWorks. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

claim-lock		Function
Summary	Set the owner o	f the specified lock to <b>*current-process*</b> .
Package	mp	
Signature	claim-lock lock	=> bool
Arguments	lock	The specified lock
Values	bool	t if the lock was successfully claimed; nil otherwise.
Description	claim-lock attempts to set the owner of the specified lock to be *current-process*, at the same time setting its lock to t, and incrementing its count.	

#### The MP Package

If the lock is successfully claimed, claim-lock returns t, otherwise it returns mil.

See also make-lock process-lock process-unlock release-lock with-lock

#### create-process

Summary	Creates a new process to be executed in multiprocessing mode.	
Package	mp	
Signature	create-process name & rest init-args &key priority run-reasons stack-group arrest-reasons &allow-other-keys => process	
Arguments	name	The identifirer of the function to be run in multiprocessing mode
	priority	A fixnum describing the priority of the new process.
	run-reasons	The reasons for the execution of the new process.
	stack-group	The stack-group of the new process.
	arrest-reasons	The reasons for the termination fo the new process.
Values	process	The new process.
Description	The create-process function computes the result of apply- ing the Lisp function <i>name</i> to supplied arguments. Argu- ments to create-process itself, which are detailed above, may be supplied in addition to <i>name</i> 's arguments.	

	A <i>priority</i> for the process may be specified. This must be a <code>fixnum</code> value. If no such value is supplied, the process priority becomes the value of the variable <code>*default-process-priority*</code> .
	This function is typically called via process-run-function, and it is recommended that process-run-function be used by programmers in preference to create-process.
See also	process-run-function

#### create-simple-process

Summary	Creates and returns a simple process, which is a process with no stack of its own.	
Signature	create-simple-process name function wait-function &key function-arguments wait-function-arguments priority => process	
Package	mp	
Arguments	name	A string or symbol
	function	A function
	wait-function	A function
	function-arguments	
		A list of arguments for function
	wait-function-arguments	
		A list of arguments for wait-function
	priority	A fixnum
Values	process	A simple process

Description The create-simple-process function creates and returns a simple process, which is a process that has no stack of its own.

The *name* argument is a string or symbol that names the process. The *function* argument is a function to be run in the process, and the *wait-function* argument is a wait function that determines when the process function is run. The value of *function-arguments* is a list of arguments to which the process function is applied. The value of *wait-function-arguments* is a list of arguments to which the wait function is applied. The *:priority* argument is a fixnum that specifies a priority for the process. The default priority is the value of \*defaultsimple-process-priority\*, and is usually 0.

When the wait function, applied to the *wait-function-arguments*, returns a value other than nil, the process function is applied to the function-arguments. The process function is executed inside an mp:without-preemption form. If an error occurs in a simple process, that process is stopped and a continuable error is signaled in the process that was running at the time the simple process was started (or the last process to run if the system was idle). Continuing from the error restarts the simple process.

Because a simple process has no stack of its own, it can be executed on an arbitrary stack. However, simple processes have restrictions, the primary one being that they cannot block. The following interfaces cannot be used in a simple process:

- mp:mailbox-read (with an empty mailbox)
- mp:process-allow-scheduling
- mp:process-lock
- mp:process-wait
- mp:process-wait-with-timeout
- cl:sleep

- ٠ mp:sleep-for-time
- mp:wait-for-mailbox
- mp:with-lock
- CAPI functions that block .

Other Common Lisp functions might not work if they attempt to block. This applies in particular to I/O functions on streams such as pipes and to (setf gethash) on a hash table that another process is mapping over.

For more information, see the "Multi-Processing" chapter of the LispWorks Users Guide.

Example The following example creates a simple process that prints the value of **\*a\*** to the background output when the value is other than nil. The process function then sets \*a\* to nil. From a listener, the value of \*a\* can be set to trigger the process to run once and then sleep again.

```
(defvar *a* 'i)
             *A*
             (defun a ()
               (let ((a *a*))
                  (setq *a* nil)
                  (format mp:*background-standard-output*
                                 "*a* is ~a~%" a)))
             А
             (defun b () *a*)
             в
             (setq r (mp:create-simple-process 'test-proc 'a 'b))
             #<MP::SIMPLE-PROCESS Name TEST-PROC Priority 0 State
             NIL>
See also
```

process-run-function

# \*current-process\* Variable Summary Contains the object that is the current process. Package mp Description This special variable contains the object that is the current process. \*default-process-priority\* Variable

Summary	The default priority for processes.
Package	mp
Description	The <b>*default-process-priority*</b> variable contains the default priority for processes.
See also	process-run-function create-simple-process *default-simple-process-priority*

# \*default-simple-process-priority\*VariableSummaryThe default priority for simple processes.

 Package
 mp

 Description
 The \*default-simple-process-priority\* variable contains the default priority for simple processes.

 See also
 create-simple-process

ee also create-simple-process \*default-process-priority\*

#### ensure-process-cleanup

Summary	Run forms when a given process terminates.		
Package	mp		
Signature	ensure-process-cleanup <i>cleanup-form</i> &optional <i>process</i> =>		
Arguments	cleanup-form	Form to run when <i>process</i> terminates.	
	process	The process to watch for termination. By default, this is the value of mp:*current-process*.	
Values	None		
Description	Ensures that the <i>cleanup-form</i> is present for the given process When the process terminates, its cleanup forms are run. Cleanup forms can be functions of one argument (the pro- cess), or lists, in which case the car is applied to the process and the cdr of the list.		
	When adding cleanup forms, this function uses equal to ensure that the form is only added once.		
Example	A process calls add-process-dependent each time a depen- dent object is added to a process. When the process termi- nates, inform-dependent-of-dead-process is called on all dependent objects.		
	<pre>(defun add-process-dependent (dependent)   (mp:ensure-process-cleanup</pre>		
		process-dependent (process dependent) ndent-of-dead-process dependent process))	
See also	process-kill		

#### find-process-from-name

Function

Finds a process based on the name of the process.
mp
<pre>find-process-from-name process-name =&gt; process</pre>
<i>process-name</i> A string, the name of a process.
<i>process</i> The process named by <i>process-name</i> .
Returns the process with the given name.
CL-USER 16 > (mp:find-process-from-name "Listener 1") # <mp:process "listener="" 1"="" 600000="" name="" priority="" state<br="">"Running"&gt;</mp:process>

#### initialize-multiprocessing

Summary	Initializes multiprocessing before use.
Package	mp
Signature	initialize-multiprocessing => nil
Arguments	None.
Values	Returns nil.
Description	The mp:process-run-function function is applied to the each of the entries in mp:*initial-processes* to create the initial processes.
	It is not necessary to call this function when the Common LispWorks is running (i.e., when env:start-environment has been called), as this automatically starts up multiprocess-

ing. Common LispWorks starts automatically as soon as you start the image.

#### \*initial-processes\*

Special Variable

Summary	A list of the processes the system initializes on startup.
Package	mp
Description	Each element of the <b>*initial-processes*</b> list is a set of arguments for <b>process-run-function</b> .

#### list-all-processes

Summary	Lists all the Lisp processes currently in the system.	
Package	mp	
Signature	list-all-processes => process-list	
Arguments	None.	
Values	<i>process-list</i> A list of all the currently active Lisp processes.	
Description	Returns a list of all the active Lisp processes in LispWorks.	
Example	<pre>NEW 38 &gt; (mp:list-all-processes) (#<mp:process "lispworks="" 4.0.0"="" 700000<br="" name="" priority="">State "Waiting for events"&gt; #<mp:process "editor="" 1"="" 600000="" name="" priority="" state<br="">"Waiting for events"&gt; #<mp:process "listener="" 1"="" 600000="" name="" priority="" state<br="">"Running"&gt; #<mp:process "listener="" 1"="" 600000="" name="" priority="" state<br="">"Running"&gt; #<mp:process "the="" -8388608<br="" idle="" name="" priority="" process"="">State "Running"&gt;)</mp:process></mp:process></mp:process></mp:process></mp:process></pre>	

The MP Package

lock-name			Function
Summary	Returns the name of a lock.		
Package	mp		
Signature	lock-name lo	ock => name	
Arguments	lock	A lock object	
Values	name	A string	
Description		me function takes a lock object a the name of the lock object.	as its argument
Example		(mp:make-lock :name "my loc name lock))	k")))
	=> "my lock"	n	
See also	make-lock with-lock process-loc process-unl lock-owner		
lock-owner			Function
Summary	Returns the o	current owner of a lock.	
Package	mp		
Signature	lock-owner	lock => owner	
Arguments	lock	A lock object	

Values owner A process

Description	The lock-owner function returns the process that currently owns the lock, or nil.		
Example	<pre>(let ((lock (mp:make-lock :name "my lock")))  (mp:lock-owner lock))</pre>		
	=> "my lock"		
	<pre>(let ((lock (mp:make-lock :name "my lock")))  (mp:with-lock (lock)  (mp:lock-owner lock)))</pre>		
	=> # <mp:process "listener="" 1"="" 700000="" name="" priority="" state<br="">"Running"&gt;</mp:process>		
See also	make-lock		
	with-lock		
	process-lock		
	process-unlock		
	lock-name		

#### mailbox-empty-p

Summary	Tests whether a	n mailbox is empty.
Package	mp	
Signature	mailbox-empty-	p mailbox => bool
Arguments	mailbox	A mailbox
Values	bool	A generalized boolean
Description	The mailbox-empty-p function returns t if the given <i>mailbox</i> is empty and mil otherwise.	
See also	mailbox-send mailbox-peek	

mailbox-read make-mailbox

#### mailbox-peek

#### Function

Summary	Peeks at the firs	st object in a mailbox.
Package	mp	
Signature	mailbox-peek mailbox => result	
Arguments	mailbox	A mailbox
Values	result	Any object or nil
Description		eek function returns the first object in the at removing it. If the mailbox is empty, nil is
See also	mailbox-empty mailbox-send mailbox-read make-mailbox	-p

#### mailbox-read

Summary	Reads the next	object in a mailbox.
Package	mp	
Signature	mailbox-read 1	nailbox & optional wait-reason => object
Arguments	mailbox	A mailbox
	wait-reason	A string

Values	object	Any object
Description	mailbox. If the r	and function returns the next object from the mailbox is empty, this function blocks until an in the mailbox. The <i>wait-reason</i> is used as the process-wait.
See also	mailbox-empty mailbox-peek mailbox-send make-mailbox	-p

#### mailbox-reader-process

#### Function

Function

Summary	Returns the read	der process of the mailbox.
Package	qm	
Signature	mailbox-reader	-process mailbox => process
Arguments	mailbox	A mailbox
Values	process	A process
Description	The mailbox-reader-process function returns the reader process of <i>mailbox</i> .	
See also	process-send	

#### mailbox-send

- Summary Sends an object to a mailbox.
- Package mp

#### The MP Package

Signature	mailbox-send $n$	nailbox object =>
Arguments	mailbox object	A mailbox An object
Description		and sends <i>object</i> to <i>mailbox</i> . The object is nailbox for retrieval by the reader.
See also	mailbox-empty mailbox-peek mailbox-read make-mailbox	-p

#### make-lock

Summary	Makes a lock.	
Package	mp	
Signature	make-lock &key	important-p name lock owner count => lock
Arguments	important-p	When this is t, the lock is pushed onto the list mp:*important-locks*. Locks in this list are automatically freed when the holder process finishes.
		This should be nil for locks which are man- aged completely by the application. It is wasteful to record all locks in a global list if there is no need to free them automatically.
		This might be appropriate when two pro- cesses sharing a lock must both be running for the system to be consistent. If one pro- cess dies, then the other one kills itself. Thus the system does not need to worry about

		freeing the lock because no-one else could be waiting on it forever after the first process dies.
	name	The name of the lock.
	lock	t or nil. When lock is t, the lock is set. The default is nil.
	owner	The owner of the lock.
	count	The number of times the lock has been set. The default is 0.
Values	lock	The lock object.
Description		rns a lock object. See the <i>Multiprocessing</i> chap- <i>Torks User Guide</i> for a general description of
Example	-	tq *my-lock* (mp:make-lock :name "my-lock")) k" "Unlocked" NIL 0 7909676)
See also	claim-lock process-lock process-unloc release-lock with-lock	k

#### make-mailbox

#### Function

SummaryMake a new mailbox for the current process.PackagempSignaturemake-mailbox &key size lock-name => mailboxArgumentssizeAn integer

#### The MP Package

	lock-name	A string
Values	mailbox	A mailbox
Description	size and the nar	<b>bx</b> function returns a new mailbox. The initial ne of the lock are specified by the <i>size</i> and ords. The reader process is set to the current
See also	mailbox-empty- mailbox-peek mailbox-read mailbox-send	-p

#### make-named-timer

Summary	Creates and returns a named timer.	
Package	mp	
Signature	make-named-tim	er name function &rest arguments => timer
Arguments	name	A string or symbol
	function	A function
	arguments	A set of arguments to <i>function</i>
Values	timer	A timer
Description	The make-named-timer function creates and returns a named timer. The first argument is a string or symbol naming the timer. The second argument is a function to be applied to the remaining arguments when the timer expires. Use the func- tion schedule-timer or schedule-timer-relative to set an expiration time.	

	In comparison, the function make-timer creates an unnamed timer.
Example	(setq timer (mp:make-named-timer 'timer-1 'print 10 *standard-output*))
	# <time :="" event="" print=""></time>
See also	<pre>make-timer schedule-timer schedule-timer-milliseconds schedule-timer-relative schedule-timer-relative-milliseconds timer-expired-p timer-name unschedule-timer</pre>

#### make-timer

Summary	Creates and ret	turns an unnamed timer.
Signature	make-timer fun	action &rest arguments => timer
Package	mp	
Arguments	function arguments	A function A set of arguments to <i>function</i>
Values	timer	A timer
Description	timer. The <i>func</i> remaining argu	r function creates and returns an unnamed tion argument is a function to be applied to the uments when the timer expires. Use the func- timer Or schedule-timer-relative to set an e.

Note that the function make-named-timer creates a named timer.

Example	(setq timer (mp:make-timer 'print 10 *standard-output*))		
	# <time :="" event="" print=""></time>		
See also	make-named-timer		
	make-timer		
	schedule-timer		
	schedule-timer-milliseconds		
	schedule-timer-relative		
	schedule-timer-relative-milliseconds		
	timer-expired-p		
	timer-name		
	unschedule-timer		

#### map-all-processes

Summary	Calls the function for all processes.
Package	mp
Signature	map-all-processes function =>
Arguments	<i>function</i> A function taking one argument
Values	None
Description	The map-all-processes function calls <i>function</i> for every process, including simple processes. The <i>function</i> is passed a single argument, namely a process.
See also	map-process

#### map-all-processes-backtrace

#### Summary Produces a backtrace for every known process. Package mp Signature map-all-processes-backtrace & optional function Arguments function A function taking one argument Values None Description The map-all-processes-backtrace function calls function, which defaults to print, for every known process and each line of its backtrace. See also map-process-backtrace

#### map-process-backtrace

Function

for

Summary	Produces a bac	ktrace for a process
Package	mp	
Signature	map-process-ba	acktrace process function
Arguments	process	A process
	function	A funtion taking one argument
Values	None	
Description	the process spe	ss-backtrace function collects a backtrace for cified by <i>process</i> , and the function given by a is print by default) is called on each line of

See also	map-all-processes-backtrace
----------	-----------------------------

#### map-process

Summary	Calls the function for all non-simple processes.
Package	mp
Signature	map-process function
Arguments	<i>function</i> A function taking one argument
Values	None
Description	The map-process function calls <i>function</i> for every non-simple process. The function is passed a single argument that is the name of a process.
See also	map-all-processes
matica fel	
notice-fd	Function
Summary	Add a file descriptor to the set of interesting input file descriptors.
Package	mp
Signature	notice-fd fd
Arguments	fd A UNIX file descriptor
Values	None
Description	The notice-fd function adds the given <i>fd</i> to the set of fds that cause LispWorks to wake up when they contain input.

See also unnotice-fd

#### \*processes\*

#### Special Variable

Package	mp
Description	This variable contains a list of all the current Lisp processes.

#### process-alive-p

Summary	Determines if a	process is alive.
Package	mp	
Signature	process-alive-	p process => bool
Arguments	process	A process
Values	bool	A boolean
Description		ive-p function returns t if <i>process</i> is alive, bcess-kill has not been called on the pro-
Example	(mp:process-al => T	<pre>ive-p mp:*current-process*)</pre>
	<pre>(let ((process (sleep 2) (mp:process- =&gt; NIL</pre>	<pre>(mp:process-run-function   "test" nil `identity nil))) alive-p process))</pre>

#### process-allow-scheduling

SummaryAllows scheduling within a process, so that the process is<br/>interruptible.PackagempSignatureprocess-allow-scheduling =>ArgumentsNoneValuesNoneDescriptionThis gives other Lisp processes a chance to run.

Function

Function

#### process-arrest-reasons

#### Summary Returns a list of the reasons why a Lisp process has stopped. Package mp Signature process-arrest-reasons process => reasons Arguments process A process. Values A list of reasons. reasons Description Returns a list of the reasons why a Lisp process has stopped. A process is inactive if it has any arrest reasons. The list can be changed using setf, though it is not usually necessary to add arrest reasons. See also process-run-reasons

#### process-break

#### Function

 Summary
 Forces the specified Lisp process to break and enter the TTY debugger.

 Package
 mp

 Signature
 process-break process =>

 Arguments
 process

 Values
 None.

#### \*process-initial-bindings\*

Special Variable

Function

Summary	Specifies the variable	s initially bound in	a new process.
---------	------------------------	----------------------	----------------

#### Package mp

Description This specifies the variables that are initially bound in a Lisp process when that process is created. This variable is an association list of symbols and initial value forms. The initial value forms are processed by a simple evaluation that handles symbols and function call forms, but not special operators.

#### process-interrupt

- Summary Interrupts a process.

Package mp

Signature process-interrupt process function &rest arguments =>

Arguments *process* A process.

#### The MP Package

	function arguments	A function to apply on resuming <i>process</i> . Arguments to supply to <i>function</i> .	
Values	None.		
Description	when it is ne	isp process <i>process</i> to apply <i>function</i> to <i>argument</i> . ext resumed. Afterwards the process resumes its ution. A waiting process is temporarily woken	
process-kil	I	Functio	n
Summary	Kills the spec	cified Lisp process.	
Package	mp		
Signature	process-kill	1 process =>	
Arguments	process	A process.	
Values	None.		
See also	ensure-proc	ess-cleanup	
process-loc	:k	Functio	n
Summary	Claims the lo	ock for the current process.	
Package	mp		
Signature	process-loc)	k lock &optional whostate timeout => bool	
Arguments	lock	A lock object (see make-lock).	

	whostate	The status of the current Lisp process, before process-lock returns, that is, the status while the current process is waiting to time-out. This can be seen in the Process Browser.
	timeout	A timeout interval, in seconds. If this is not given, process-lock waits until the lock can be set by the current Lisp process. A process can set a lock more than once.
Values	bool	t if the process was successfully locked; נות otherwise.
Description	ful, or nil if tim lock is the value is incremented	attempts to set a lock and returns $t$ if success- ned out. If the lock is set and the owner of the e of *current-process*, then the lock's count by 1, and the lock is set to t. The Lisp process lock is claimed or the timeout period expires.
Example	(process-lock	*my-lock* "waiting to lock" 10)
See also	claim-lock make-lock process-unloc release-lock with-lock	k
process-name	è	Function
Summary	Returns the nar	ne of a specified process.
Package	mp	
Signature	process-name p	rocess => name
Arguments	process	A process.

Values *name* The name of the process specified by *process*.

Description Returns the name of the specified Lisp process.

#### Function process-p Summary A predicate to indentify non-simple processes Package mp Signature process-p object => bool Arguments object Any object Values bool A generalized boolean. Description The process-p function returns t if *object* is a non-simple process, and nil otherwise. Function process-plist Summary Returns the plist associated with a process.

Package	mp	
Signature	process-plist	process => plist
Arguments	process	A process
Values	plist	A plist
Description	The process-p: process.	List function returns the plist associated with
Example		p:process-plist mp:*current-process*) foo-value)
	=> FOO-VALUE	

```
(getf (mp:process-plits mp:*current-process*) `foo)
=> FOO-VALUE
```

#### process-priority

#### Function

Summary	Returns the nun	nerical priority of the Lisp process.
Package	mp	
Signature	process-priori	ty process => priority
Arguments	process	A process.
Values	priority	The priority of <i>process</i> , as a number.
Description		nerical priority of the Lisp process. This can calling mp:change-process-priority.
Example	CL-USER 17 > (1 600000	<pre>mp:process-priority mp:*current-process*)</pre>

#### process-reset

Summary	Resets a process by discarding its current state.
Package	mp
Signature	<pre>process-reset process =&gt;</pre>
Arguments	process A process.
Values	None.
Description	process-reset interrupts the execution of process and "throws away" its current state. Upon resuming execution,

the process calls its function with its initial argument and priority.

#### process-run-function

#### Function

Summary	Create a new process, passing it a function to run.	
Package	mp	
Signature	<pre>process-run-function name keywords function &amp;rest arguments =&gt; process</pre>	
Arguments	name	A name for the new process.
	keywords	Keywords to be passed to the new process. See create-process for details.
	function	A function to apply.
	arguments	Arguments to pass to function.
Values	process	The newly created process.
Description	This function creates a new Lisp process, passing it <i>name</i> (a string) and <i>keywords</i> . The new process is preset to apply <i>func-tion</i> to <i>arguments</i> and runs in parallel, while process-run-function returns immediately.	
See also	create-simple-	process

#### process-run-reasons

Summary	Returns the reasons that a specified process is running.
Package	mp
Signature	process-run-reasons process => reasons

Arguments	process	A process.
Values	reasons	A list of run reasons.
Description	ning. These can	f reasons for the specified Lisp process run- be changed using setf. A process is only at least one run reason and no arrest reasons.
See also	process-arres process-run-f	

## process-send

Summary	Sends an object to the mailbox of a given process.	
Package	mp	
Signature	process-send process thing &key change-priority =>	
Arguments	processA processobjectAn objectchange-priorityA fixnum, nil, t, or :default	
Values	None	
Description	The process-send function queues object in the mailbox of the given process. If <i>change-priority</i> , which has a default value of :default, is non-nil, it controls how the priority of that process is calculated as follows:	
	• fixnum — use the value of <i>change-priority</i> as the new priority.	
	• $t$ — set the priority to the interactive priority.	
	• <pre>:default — set the priority to the normal running prior- ity.</pre>	

See also mailbox-reader-process mailbox-send

## process-unlock

### Function

Summary	Relinquishes a lock held by the current process.	
Package	mp	
Signature	process-unlock	lock & optional errorp => bool
Arguments	lock errorp	The lock to be relinquished. When this is t, an error is signalled if *current-process* is not the owner of the lock. The default is t.
Values	bool	t if the lock was unset; nil otherwise.
Description	Attempts to unset a lock. If the lock is owned by *current-process*, process-unlock decrements the lock count by 1. If the lock count is now nil, the lock is released. Note that process-unlock relates only on Lisp processes.	
See also	claim-lock make-lock process-lock release-lock with-lock	

## process-wait

### Function

Summary	Suspend the current process until certain conditions are true.

Package mp

Signature	process-wait wait-reason wait-function &rest wait-arguments =>	
Arguments	<i>wait-reason</i> A reason that the process is waiting. This is a string.	
	wait-function	A function to test.
	wait-arguments	The arguments to apply to wait-function.
Values	None.	
Description	This function suspends the current Lisp process until the predicate <i>wait-function</i> applied to <i>wait-arguments</i> returns t. This is tested periodically. <i>wait-reason</i> (a string) allows you to find out why a process is waiting.	
See also	process-wait-with-timeout	

## process-wait-function

Summary	Returns a funct cess is waiting.	ion that gives a reason why the specified pro-
Package	mp	
Signature	process-wait-f	unction process => wait-function
Arguments	process	A process.
Values	wait-function	A function specifying a wait reason for <i>pro-</i> cess.
Description	Returns a function that specifies a reason for the Lisp process waiting. The system periodically applies process-wait- function to the <i>wait-arguments</i> given to process-wait to decide whether to wake the process up. A process is gener- ally given a wait function by means of process-wait.	

See also process-wait

## process-wait-with-timeout

### Function

Suspend the current process until certain conditions are true, or until a timeout expires.		
mp		
process-wait-with-timeout wait-reason timeout &optional wait-function &rest wait-arguments => bool		
wait-reason	A reason that the process is waiting. This is a string.	
timeout	A timeout, in seconds.	
wait-function	A function to test.	
wait-arguments	The arguments to apply to <i>wait-function</i> .	
bool	A boolean.	
This function uses process-wait to suspend the current Lisp process until the predicate <i>wait-function</i> applied to <i>wait-arguments</i> returns t, or until <i>timeout</i> seconds have passed.		
process-wait		
	Function	
Prints the proce	esses in the system	
	or until a timeo mp process-wait-w &optional wait-reason timeout wait-function wait-arguments bool This function us process until th ments returns t,	

Signature ps =>

Arguments	None.
Values	None.
Description	Prints a list of the processes in the system, ordered by priority. (This function is analogous to the UNIX command $ps$ .)

## release-lock

Summary	Releases a lock.	
Package	mp	
Signature	release-lock $la$	ock =>
Arguments	lock	The lock to be released
Values	None.	
Description	Completely rele ni1, and count t	ases a lock: the owner is set to mil, the lock to to 0.
See also	claim-lock make-lock process-lock process-unlock with-lock	τ

### schedule-timer

### Function

Function

 Summary
 Schedules a timer to expire at a given time after the start of the program.

 Signature
 schedule-timer timer absolute-expiration-time & optional repeat-time => timer

Package	mp		
Arguments	timer	A timer	
	absolute-expiration	on-time	
		A non-negative real	
	repeat-time	A non-negative real	
Values	timer	A timer	
Description	The schedule-timer function schedules a timer to expire at a given time after the start of the program. The <i>timer</i> argument is a timer, returned by make-timer or make-named-timer. The <i>absolute-expiration-time</i> argument is a non-negative real number of seconds since the start of the program at which the timer is to expire. If <i>repeat-time</i> is specified, it is a non-negative real number of seconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.		
	function is calle fied by the <i>absol</i> is nil, the timer	ready scheduled to expire at the time this d, it is rescheduled to expire at the time speci- <i>ute-expiration-time</i> argument. If that argument is not rescheduled, but the repeat interval is al specified by the <i>repeat-time</i> argument.	
		hedule-timer-relative schedules a timer to relative to the call to that function.	
Example	The following example schedules a timer to expire 15 min- utes after the start of the program and every 5 minutes there- after.		
	(setq timer (mp:make-timer 'print 10 *standard-output*))		
	<pre>#<time :<="" event="" pre=""></time></pre>	PRINT>	
	(mp:schedule-timer timer 900 300)		
	# <time :="" event="" print=""></time>		

See also	make-named-timer	
	make-timer	
	schedule-timer-milliseconds	
	schedule-timer-relative	
	schedule-timer-relative-milliseconds	
	timer-expired-p	
	timer-name	
	unschedule-timer	

## schedule-timer-milliseconds

Summary	Schedules a timer to expire after a given amount of time.	
Signature	schedule-timer-milliseconds <i>timer absolute-expiration-time</i> &optional <i>repeat-time =&gt; timer</i>	
Package	mp	
Arguments	timer	A timer
	absolute-expirati	on-time
	A non-negative real	
	repeat-time	A non-negative real
Values	timer	A timer
Description	The schedule-timer-milliseconds function schedules a timer to expire at a given time after the start of the program. The timer argument is a timer returned by make-timer or make-named-timer. The absolute-expiration-time argument is a non-negative real number of milliseconds since the start of the program at which the timer is to expire. If repeat-time is specified, it is a non-negative real number of milliseconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.	

	If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time specified by the <i>absolute-expiration-time</i> argument. If that argument is nil, the timer is not rescheduled, but the repeat interval is set to the interval specified by the <i>repeat-time</i> argument.	
	The function schedule-timer-relative-milliseconds schedules a timer to expire at a time relative to the call to that function.	
Example	The following example schedules a timer to expire 15 min- utes after the start of the program and every 5 minutes there- after.	
	(setq timer (mp:make-timer 'print 10 *standard-output*))	
	# <time :="" event="" print=""></time>	
	(mp:schedule-timer-milliseconds timer 900000 300000)	
	# <time :="" event="" print=""></time>	
See also	<pre>make-named-timer make-timer schedule-timer schedule-timer-relative schedule-timer-relative-milliseconds timer-expired-p timer-name unschedule-timer</pre>	

## schedule-timer-relative

Summary	Schedules a timer to expire at a given time after this function is called.
Signature	<pre>schedule-timer-relative timer relative-expiration-time &amp;optional repeat-time =&gt; timer</pre>

Package	mp		
Arguments	timer	A timer	
	relative-expiration	on-time	
		A non-negative real	
	repeat-time	A non-negative real	
Values	timer	A timer	
Description	The schedule-timer-relative function schedules a timer to expire at a given time after the call to the function. The <i>timer</i> argument is a timer returned by make-timer or make-named- timer. The <i>relative-expiration-time</i> argument is a non-negative real number of seconds after the call to the function at which the timer is to expire. If <i>repeat-time</i> is specified, it is a non-neg- ative real number of seconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.		
	If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time spec fied by the <i>relative-expiration-time</i> argument. If that argument is nil, the timer is not rescheduled, but the repeat interval set to the interval specified by the <i>repeat-time</i> argument.		
		thedule-timer schedules a timer to expire at a the start of the program.	
Example	The following example schedules a timer to expire 5 seconds after the call to schedule-timer-relative and every 5 seconds thereafter.		
	(setq timer (mp:make-timer 'print 10 *standard-output*))		
	# <time :="" event="" print=""></time>		
	(mp:schedule-t	imer-relative timer 5 5)	
	# <time :="" event="" print=""></time>		

See also	make-named-timer, make-timer, schedule-timer, schedule-
	timer-milliseconds, schedule-timer-relative-millisec-
	onds, timer-expired-p, timer-name, unschedule-timer

Function

### schedule-timer-relative-milliseconds

Summary Schedules a timer to expire at a given time after this function is called. Signature schedule-timer-relative-milliseconds timer relative-expiration-time &optional repeat-time => timer Package mp Arguments A timer timer relative-expiration-time A non-negative real repeat-time A non-negative real Values timer A timer Description The schedule-timer-relative-milliseconds function schedules a timer to expire at a given time after the call to the function. The *timer* argument is a timer returned by maketimer Or make-named-timer. The relative-expiration-time argument is a non-negative real number of milliseconds after the call to the function at which the timer is to expire. If *repeattime* is specified, it is a non-negative real number of milliseconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval. If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time specified by the *relative-expiration-time* argument. If that argument is nil, the timer is not rescheduled, but the repeat interval is

set to the interval specified by the *repeat-time* argument.

	The function schedule-timer-milliseconds schedules a timer to expire at a time relative to the start of the program.			
Example	The following example schedules a timer to expire 5 seconds after the call to schedule-timer-relative-milliseconds and every 5 seconds thereafter.			
	(setq timer (mp:make-timer 'print 10 *standard-output*))			
	# <time :="" event="" print=""></time>			
	(mp:schedule-timer-relative-milliseconds timer 5000 5000)			
	# <time :="" event="" print=""></time>			
See also	make-named-timer			
	make-timer			
	schedule-timer			
	schedule-timer-milliseconds			
	schedule-timer-relative			
	timer-expired-p			
	timer-name			
	unschedule-timer			

## setf timer-name

Summary	Sets the name o	f a specified timer.
Package	mp	
Signature	setf timer-nam	e name timer => timer-name
Arguments	timer	A timer to be named.
	name	The new name for <i>timer</i> .
Values	timer-name	The new name of <i>timer</i> .

The MP Package

- Description Set the *name* of the given *timer* and return the new name.
- See also timer-name

## simple-process-p

### Function

Summary	A predicate identifying simple processes.		
Package	mp		
Signature	simple-proces	<pre>simple-process-p object=&gt; bool</pre>	
Arguments	object	An object	
Values	bool	A generalized boolean	
Description		The simple-process-p function returns t if <i>object</i> is a simple process and nil otherwise.	
See also	create-simple-process		

## timer-expired-p

Summary	Returns $t$ if a given timer has expired or is about to expire	
Signature	timer-expired-	-p timer &optional delta => bool
Package	mp	
Arguments	timer delta	A timer A non-negative real
Values	bool	A boolean

Description	The timer-expired-p function returns t if the specified timer is not scheduled to expire or is scheduled to expire within the number of seconds specified by the <i>delta</i> argument after the call to timer-expired-p. Otherwise, the function returns nil.	
	The <i>timer</i> argument is a timer, returned by make-timer or make-named-timer. The <i>delta</i> argument, if supplied, is a non-negative real number of seconds.	
Example	(setq timer (mp:make-timer 'print 10 *standard-output*))	
	# <time :="" event="" print=""></time>	
	(mp:schedule-timer-relative timer 5)	
	# <time :="" event="" print=""></time>	
	(mp:timer-expired-p timer)	
	NIL	
See also	make-named-timer, make-timer, schedule-timer, schedule- timer-milliseconds, schedule-timer-relative, timer- name, unschedule-timer	

## timer-name

- SummaryReturns the name of a specified timer.Signaturetimer-name timer => namePackagempArgumentstimerA timer
- Values name A string

Description	The timer-name function returns the name of the specified <i>timer</i> . The <i>timer</i> argument is a timer returned by make-timer or make-named-timer. If the timer has no name, timer-name returns nil.
	The name of a timer created by either make-timer or make- named-timer can be set by means of the following syntax:
	(setf (mp:timer-name timer) name)
Example	(setq timer (mp:make-timer 'print 10 *standard-output*))
	# <time :="" event="" print=""></time>
	(mp:timer-name timer)
	NIL
	(setf (mp:timer-name timer) 'timer-1)
	TIMER-1
	(mp:timer-name timer)
	TIMER-1
See also	make-named-timer, make-timer, schedule-timer, schedule- timer-milliseconds, schedule-timer-relative, timer- expired-p, unschedule-timer
unnotice-fd	Function
Summary	Removes a file descriptor from the set of interesting input file descriptors.
Package	mp

Signature unnotice-fd fd

ArgumentsfdA file descriptor

Values	None
Description	The unnotice-fd function removes <i>fd</i> from the set of fds that cause LispWorks to
See also	notice-fd

## unschedule-timer

Summary	Unschedules a scheduled timer		
Signature	unschedule-timer timer => result		
Package	mp		
Arguments	timer	A timer	
Values	result	A timer or mil	
Description	after the call to the timer and returns nil.	timer has been scheduled to expire at a time unschedule-timer, this function unschedules eturns the timer. Otherwise, the function s a timer, returned by make-timer Or make-	
	named-timer.	s a time, returned by make-timer of make-	
Example	(setq timer (mp:make	-timer 'print 10 *standard-output*))	
	<pre>#<time :<="" event="" pre=""></time></pre>	PRINT>	
	(mp:schedule-t	imer-relative timer 60)	
	<pre>#<time :<="" event="" pre=""></time></pre>	PRINT>	
	(mp:unschedule	-timer timer)	
	# <time :<="" event="" td=""><td>PRINT&gt;</td></time>	PRINT>	
	(mp:timer-expi	red-p timer)	

т

See also	make-named-timer, make-timer, schedule-timer, schedule-
	timer-milliseconds, schedule-timer-relative, timer-
	expired-p, timer-name

## with-lock

Macro

Summary	Executes a body	y of code while holding a lock.
Package	mp	
Signature	with-lock ( <i>loc</i> )	k &rest lock-args) &body body => result
Arguments	lock	The lock.
	lock-args	These are the optional arguments used by process-lock: <i>whostate</i> (the status of the process while the lock is set, as seen in the Process Browser) and <i>timeout</i> (a timeout period, in seconds).
	body	The forms to execute.
Values	result	The result of executing <i>body</i> .
Description	with-lock executes <i>body</i> while holding the lock, and unlocks the lock when <i>body</i> exits. This is the recommended way of using locks. The value of <i>body</i> is returned normally. <i>body</i> is not executed if the lock could not be claimed, in which case, with-lock returns nil.	
See also	claim-lock make-lock process-lock process-unloc release-lock	k

## without-interrupts

Summary	Causes any interrupts that occur during the execution of a body of code to be queued.		
Package	mp		
Signature	without-interr	upts &rest body => result	
Arguments	body	The forms to execute while interrupts are queued.	
Values	result	The result of executing <i>body</i> .	
Description	0	ecuting, all interrupts (for example, preemp- preak etc.) are queued. They are executed	
Example	consistent, you function which		
	(defstruct ela seconds milliseconds	-	
	(elapsed- (mp:without- (setf (elap (elap)	elapsed-time-atomically time seconds milliseconds) interrupts sed-time-seconds elapsed-time) seconds sed-time-milliseconds elapsed-time) seconds)))	
See also	without-preem	ption	

## without-preemption

Macro

Summary Identifies forms which should not be preempted during execution. The MP Package

Package	mp	
Signature	without-preemp	otion &rest body => result
Arguments	body	The forms to be evaluated atomically.
Values	result	The result of executing <i>body</i> .
Description	Identifies forms cution.	s which should not be preempted during exe-
yield		Function
Summary	Allows preemp	tion to happen in low safety code.
Package	mp	
Signature	yield	
Arguments	None	
Values	None	
Description	because the new come by calling need to call this package because example if you with no function cess-allow-sc	compiled at safety 0 cannot be preempted ressary checks are ommited. This can be over- gyield at regular intervals. Usually there is no if you use functions from the common-lisp re these are not compiled at safety 0, but for find that preemption is not working in a loop n calls, yield can be useful. Note that pro- heduling also allows preemption, but also functions of other processes.
See also	process-allow	-scheduling

# The PARSERGEN Package

This chapter describes symbols available in the **PARSERGEN** package, the Lisp-Works parser generator. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

### defparser

Macro

Summary	Creates a parsing function of the given name for the grammar defined.
Package	parsergen
Signature	<pre>defparser name {rule}* =&gt; parsing-function rule ::= normal-rule   error-rule normal-rule ::= ((non-terminal {grammar-symbol}*) {form}*) error-rule ::= ((non-terminal :error) {form}*)</pre>
Arguments	nameThe name of the parser.The rules define the productions of the grammar and the associated forms define the semantic actions for the rules.
Values	parsing-function The symbol name of the parsing function.

Description defparser creates a parsing function of the given name for the grammar defined. The parsing function is defined as if by:

```
(defun <name> (lexer &optional (symbol-to-string
#'identify))
```

The *lexer* parameter is a function of no arguments that returns two values: the next grammar token on the input and the associated semantic value.

The optional symbol-to-string function can be used to define a printed representation of the grammar tokens. The function should take a grammar symbol as its single argument and returns an object to be used as a print representation for the grammar token.

For a full description and examples, see the *LispWorks User Guide*.

## 14

# The SCM Package

This chapter describes symbols available in the scm package that control aspects of source control management.

### concatenate-system

Summary	This function pr of individual sy	roduces a single, concatenated fasl from a list rstems.
Package	scm	
Signature	concatenate-sy	stem output-file &rest args => nil
Arguments	In the following list of arguments, the words in parentheses are <i>equivalent forms for the keyword</i> :	
	output-file	The name of the required concatenated fasl.
	simulate	Verbosity conditions, see Description for more detail. This argument can be abbrevi- ated as <i>sim</i> .

	force	If t, then all files in the system are concate- nated. The argument <i>force-p</i> can also be used.	
	source-only	If $t$ , the source files of the system are concatenated. The argument <i>source-p</i> can also be used.	
	target-directory	The directory to search for the object files. This argument can be abbreviated as <i>t-dir</i> .	
	target-machine	Architecture of machine. This argument can be abbreviated as <i>t-machine</i> .	
Values	Returns nil.		
Description	Since concatenated fasl files may be produced using this function, you do not need to be wary of PC filename conven- tions if developing sources on UNIX for a PC application. This clearly allows more freedom for naming source files. However, <i>output-file</i> must, in such cases, be a PC-compatible filename.		
	If <i>simulate</i> is nil or is not present, concatenate-system works silently. Otherwise, a plan of the actions which con- catenate-system intends to carry out is printed. What hap- pens next depends upon the value of <i>simulate</i> :		
	• If it is t, the function does nothing.		
	<ul> <li>If :ask, then you are asked, using y-or-n-p, if the plan should be carried out.</li> </ul>		
	• If it is :each, you are asked at each stage in the plan if the current action should be carried out. The responses y and n work as normal. If e is typed, concatenate-system exits without further processing.		
	If <i>source-only</i> is t, files are loaded only if they are sources.		

If, when searching *target-directory* for an object file, the file cannot be found, the appropriate source file from the system's default directory is loaded instead.

The *target-machine* argument allows the concatenation of fasl files for loading on another machine. Possible values are:

```
:harlequin-pc-lisp OF :pc
```

Concatenate PC LispWorks fasl files.

```
:harlequin-unix-lisp Or :unix
```

Concatenate fasl files for the current UNIX machine.

By default, the concatenated fasl is built for the current machine architecture.

The SCM Package

## 15

# The SQL Package

This chapter describes the symbols available in the sql package. This package is only applicable to the Enterprise version of LispWorks. Symbols from this package can be used to provide your Common Lisp applications with SQL functionality. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

### add-sql-stream

Summary	Adds a new stre mand or result	eam to the broadcast stream for SQL com- traffic.
Package	sql	
Signature	add-sql-stream	stream &key type database => new-stream
Arguments	type stream database	A keyword. The broadcast stream. A database.
Values	new-stream	The new stream.

Description	The add-sql-stream function adds a new stream to the broadcast stream for SQL command or result traffic.
	The argument <i>type</i> is one of :command or :result, and determines whether a stream for command or result traffic is added.
	The argument <i>type</i> has a default value of :command. The <i>data-base</i> is *default-database* by default.
See also	delete-sql-stream list-sql-streams sql-recording-p sql-stream start-sql-recording stop-sql-recording

## attribute-type

Summary	Returns the type	e of a table's attribute.
Package	sql	
Signature	attribute-type	attribute table &key database => datatype
Arguments	table attribute database	A table. An attribute from <i>table</i> . A database.
Values	datatype	A type.
Description	The function attribute-type returns the type of the attribute specified by <i>attribute</i> in the table given by <i>table</i> . The database, in which <i>table</i> is found, has a default value of *default-database*.	

Example	This example shows how to query the type of the name attribute of the employee table.
	(attribute-type [ename] [emp]) => ("CHAR")
See also	list-attributes
commit	Function
Summary	Commits changes made to a database.
Package	sql
Signature	commit &key database => nil
Arguments	database A database.
Values	nil
Description	The commit function commits changes made to the database specified by <i>database</i> , which is <b>*default-database</b> * by default.
Example	<pre>In the next example, changes records in a database, and com- mit is used to make those changes permanent. (insert-records :into [emp] :attributes '(x y z) :values '(a b c)) (update-records [emp] :attributes [dept] :values 50 :where [= [dept] 40]) (delete-records :from [emp] :where [&gt; [salary] 300000]) (commit)</pre>
See also	rollback with-transaction

connect		Function
Summary	Opens a connec	ction to a database.
Package	sql	
Signature	connect connect	ion-spec &key if-exists database-type => database
Arguments	connection-spec	The connection specifications.
	if-exists	A keyword.
	database-type	A database type.
Values	database	A database.
Description	The connect function opens a connection to a database of <i>database-type</i> . The default value for <i>database-type</i> is *default-database-type*. The <i>connection-spec</i> depends on the type of the database to connect to. For databases of type :odbc, the connection specification is a string of the format	
	"datasource-name/username/password"	
	<ul> <li>where datasource-name is the name of an ODBC datasource, and username and password are a valid username and password. If the datasource does not require a username and password they can be omitted.</li> <li>The variable *default-database* is set to an instance of the database opened and the instance is returned. The argument <i>if-exists</i> modifies the behavior of connect as follows:</li> </ul>	
	:new	Makes a new connection even if connections to the same database already exist.
	:warn-new	Makes a new connection but warns about existing connections.
	:error	Makes a new connection but signals an error for existing connections.

	:warn-old	Selects old connection if one exists (and warns) or makes a new one.
	:old	Selects old connection if one exists or makes a new one.
	The defau	lt value of <i>if-exists</i> is the value of
*connect-if-exists*.		
Example	The following example connects LispWorks to the info data- base.	
	(connect "info	")
	-	ole connects to the ODBC database personnel ame "admin" and the password "secret".
	(connect "pers	connel/admin/secret" :database-type :odbc)
See also	disconnect status find-database database-name connected-dat	abases

### \*connect-if-exists\*

#### Variable

- Summary The default value for the *if-exists* keyword of the connect function.
- Package sql
- Initial Value :error
- Description The variable \*connect-if-exists\* is the default value for the *if-exists* keyword of the connect function. It can take the following values:

:new	Instructs connect to make a new connection even if connections to the same database already exist.
:warn-new	Instructs connect to make a new connection but warn about existing connections.
:error	Instructs connect to make a new connection but signal an error for existing connections.
:warn-old	Instructs connect to select an old connection if one exists (and warns) or make a new one.
:old	Instructs connect to select an old connection if one exists or make a new one.

See also connect

## connected-databases

Summary	Returns a list of connected databases.		
Package	sql		
Signature	connected-databases => database-list		
Arguments	None.		
Values	database-list A list of connected databases.		
Description	The connected-databases function returns a list of the data- bases LispWorks is connected to.		
See also	connect		
	disconnect status		
	find-database		
	database-name		

## create-index

### Function

Summary	Creates an index for a table.	
Package	sql	
Signature	create-index name &key on unique attributes database =>	
Arguments	name on unique attributes database	The name of the index. The name of a table. A boolean. A list of attributes. A database.
Values	None.	A database.
Description	The create-index function creates an index called <i>name</i> on the table specified by <i>on</i> . The attributes of the table to index are given by <i>attributes</i> . Setting <i>unique</i> to t includes UNIQUE in the SQL index command, specifying that the columns indexed must contain unique values. The default value of <i>unique</i> is nil. The default value of <i>data- base</i> is *default-database*.	
Example	(create-index [manager] :on [emp] :unique t :attributes `([ename] [sal]))	
See also	drop-index create-table	

### create-table

Function

Summary Creates a table.

	Package	sql	
	Signature	create-table name description &key database =>	
	Arguments	nameThe name of the table.descriptionThe table properties.databaseA database.	
	Values	None.	
	Description	The create-table function creates a table called <i>name</i> and defines its columns and other properties with <i>description</i> . The argument <i>description</i> is a list containing lists of attribute-name and type information pairs.	
		The default value of <i>database</i> is <b>*default-database</b> *.	
	Example	The following code: (create-table [manager] '(([id] (char 10) not-null) ([salary] (number 8 2))))	
		is equivalent to the following SQL:	
		CREATE TABLE MANAGER (ID CHAR(10) NOT NULL,SALARY NUMBER(8,2))	
	See also	drop-table	
(	create-view	Function	
	Summary	Creates a view using a specified query.	

Signature	<pre>create-view name &amp;key as column-list with-check-option database =&gt;</pre>	
Package	sql	
Summary	Creates a view using a specified query.	

Arguments	name	The view to be created.	
	as	An SQL query statement.	
	column-list	A list.	
	with-check-optio	neck-option	
		A boolean.	
	database	A database.	
Values	None.		
Description	The create-view function creates a view called <i>name</i> using the <i>as</i> query and the optional <i>column-list</i> and <i>with-check- option</i> . The <i>column-list</i> argument is a list of columns to add to the view. The <i>with-check-option</i> adds wITH CHECK OPTION to the resulting SQL.		
		ue of <i>with-check-option</i> is nil. The default W is *default-database*.	
Example	This example creates the view manager with the records in the employee table whose department is 50.		
	(create-view [	manager] :as [select [*] :from [emp] :where [= [dept] 50]])	
See also	create-index create-table drop-view		

### create-view-from-class

Summary Creates a view in a database based on a class that defines the view.

Package sq1

Function

Signature	create-view-from-class <i>class</i> &key <i>database</i> =>	
Arguments	class database	A class. A database.
Values	None.	
Description	The function create-view-from-class creates a view in <i>data-base</i> based on <i>class</i> which defines the view. The argument <i>database</i> has a default value of *default-database*.	
See also	drop-view-from-class create-view	

### database-name

#### Returns the name of a database. Summary Package sql Signature database-name database => connection Arguments database A database. Values connection A string. Description The database-name function returns the name of the database specified by database. See also connect disconnect connected-databases find-database status

### \*default-database\*

Variable

Summary	Specifies the default database to be used for database opera- tions.
Package	sql
Initial Value	nil
Description	The variable *default-database* is set by connect and specifies the default database to be used for database opera- tions.
See also	connect

## \*default-database-type\*

Summary	Specifies the default type of database.	
Package	sql	
Initial Value	nil	
Description	The variable *default-database-type* specifies the default type of database. You can set this or it is initialized by the initialize-database-type function. It should be set to :odbc in LispWorks for the Windows operating system.	
See also	initialize-database-type	

### def-view-class

Macro

Summary Extends the syntax of defclass to allow special slots to be mapped onto the attributes of database views.

Package	sql	
Signature	def-view-class class	s name superclasses slots &rest class-options =>
Arguments	name	A class name.
	superclasses	The superclasses of the class to be created.
	slots	The slot definitions of the new class.
	class-options	The class options of the new class.
Values	class	The defined class.
Slot Options	<ul> <li>The slot options for def-view-class are :db-kind and : info. In addition the slot option :type is treated specially view classes.</li> <li>:db-kind may be one of :base, :key, :join, or :virtual, default is :base. Each value is described below:</li> </ul>	
	:base	This indicates that this slot corresponds to an ordinary attribute of the database view. You can name the database attribute by using the keyword :column. By default, the database attribute is named by the slot.
	:key	This indicates that this slot corresponds to part of the unique key for this view. A :key slot is also a :base slot.
	:join	This indicates that this slot corresponds to a join. A slot of this type will contain view class objects.
	:virtual	This indicates that this slot is an ordinary CLOS slot not associated with a database column.

A join is defined by the slot option :db-info, which takes a list. Items in the list may be:

#### :join-class class-name

This is the class to join on.

#### :home-key slot-name

This is the slot of the defining class to be a subject for the join. The argument *slot-name* may be an element or a list of elements, where elements can be symbols, nil, strings, integers or floats.

#### :foreign-key slot-name

This is the name of the slot of the :joinclass to be a subject for the join. The *slotname* may be an element or a list of elements, where elements can be symbols, nil, strings, integers or floats.

#### :target-slot *slot-name*

This is the name of a : join slot in : joinclass. This is optional and is only specified if you want the defining slot to be filled with instances of this target-slot as opposed to those of : join-class. An example of its usage is when the : join-class is an intermediate class and you are really only interested in it as a route to the :target slot.

#### :retrieval :deferred

Defers filling this slot from the database until the slot itself is accessed.

#### :retrieval :immediate

This is the opposite of :deferred and generates the join SQL for this slot whenever a query is generated on the class. In other words, this is an intermediate class only, which is present for the purpose of joining

		two entities of other classes together. The :immediate specification for :retrieval also defines :set nil.
	:set t	The slot will contain a list of pairs of instances, one of which is the target instance (if specified) and one is the join instance.
	:set nil	The slot will contain a single instance.
	object from a joi the values from foreign-key, in c if the element in name and the va	home-key and :foreign-key means that an n class will only be included in the join slot if the home-key are equal to the values in the order. These values are calculated as follows: n the list is a symbol it is taken to be a slot alue of the slot is used, otherwise the element e value. See the second example below.
	There is a need because of the tr are provided for example, a :typ	option is treated specially for view-classes. for stringent type-checking in view-classes ranslation into database data. Some methods r type checking and type conversion. For be specifier of (string 10) in SQL terms character type value with length of less than
Class Options	the name of the	or def-view-class is :base-table, which is table corresponding to the class. If you do defaults to the name of the class.
Description	maps onto a dat extends the synt mapped onto th gle tables). Whe submitted, then and the slots in	view-class creates a class called <i>name</i> which tabase view. The macro def-view-class tax of defclass to allow special <i>base slots</i> to be the attributes of database views (presently sin- en a select query that names a view-class is the corresponding database view is queried, the resulting view-class instances are filled alues from the database.

One of the default classes for *superclasses* is standard-db-object.

Examples The following example shows a class corresponding to the traditional employees table, with the employee's department given by a join with the departments table.

```
(def-view-class employee (standard-db-object)
   ((employee-number :db-kind :key
                     :column empno
                     :type integer)
    (employee-name :db-kind :base
                   :column ename
                   :type (string 20)
                   :accessor employee-name)
    (employee-department :db-kind :base
                         :column deptno
                         :type integer
                         :accessor employee-department)
    (employee-job :db-kind :base
                  :column job
                  :type (string 9))
    (employee-manager :db-kind :base
                      :column mgr
                      :type integer)
    (employee-location :db-kind :join
                       :db-info (:join-class department
                                  :retrieval :deferred
                                  :set nil
                                  :home-key
                                     employee-department
                                  :foreign-key
                                     department-number
                                  :target-slot
                                     department-loc)
                        :accessor employee-location))
                (:base-table emp))
```

The following example illustrates how lists of elements can follow :home-key and :foreign-key in the :db-info slot option.

```
(def-view-class flex-class ()
  ((schema-name :type (string 8) :db-kind :key
                                 :column schema name)
                 :type (string 32) :db-kind :key)
   (name
   (base-name :type (string 64) :column base_name)
   (super-classes :db-kind :join
                   :db-info (:home-key
                              (schema-name name)
                             :foreign-key
                              (schema-name class-name)
                             :join-class
                              flex-superclass
                             :retrieval :deferred))
   (schema :db-kind :join
            :db-info (:home-key schema-name
                      :foreign-key name
                      :join-class flex-schema
                      :set nil))
   (properties :db-kind :join
          :db-info (:home-key (schema-name name "")
                    :foreign-key
                     (schema-name class-name slot-name)
                    : join-class flex-property
                    :retrieval :deferred)))
  (:base-table flex class))
(def-view-class flex-slot ()
  ((schema-name :type (string 8) :db-kind :key
                 :column schema name)
   (class-name :type (string 32) :db-kind :key
                 :column class_name)
                 :type (string 32) :db-kind :key)
   (name
   (class :db-kind :join
           :db-info (:home-key (schema-name class-name)
                     :foreign-key (schema-name name)
                     :join-class flex-class
                     :set nil))
   (properties :db-kind :join
         :db-info (:home-key
                     (schema-name class-name name)
                   :foreign-key
                     (schema-name class-name slot-name)
                   :join-class flex-property
                   :retrieval :deferred)))
  (:base-table flex_slot))
```

```
(def-view-class flex-property ()
                ((schema-name :type (string 8) :db-kind :key
                              :column schema name)
                 (class-name :type (string 32) :db-kind :key
                              :column class_name)
                 (slot-name
                              :type (string 32) :db-kind :key
                              :column slot_name)
                 (property
                              :type (string 32) :db-kind :key)
                 (values :db-kind :join
                   :db-info (:home-key
                              (schema-name class-name
                               slot-name property)
                              :foreign-key
                              (schema-name class-name
                               slot-name property)
                                    :join-class flex-property-value
                                   :retrieval :deferred)))
                (:base-table flex_property))
             (def-view-class flex-property-value ()
                ((schema-name :type (string 8) :db-kind :key
                              :column schema name)
                 (class-name :type (string 32) :db-kind :key
                              :column class name)
                 (slot-name
                              :type (string 32) :column slot_name)
                              :type (string 32) :db-kind :key)
                 (property
                 (order
                              :type integer)
                 (value
                              :type (string 128)))
                (:base-table flex_property_value))
See also
             create-view-from-class
```

drop-view-from-class

#### delete-instance-records

Generic function

SummaryDeletes records from a database.PackagesqlSignaturedelete-instance-records instance =>ArgumentsinstanceAn instance of a view class.

Values	None.
Description	The delete-instance-records function deletes the records represented by <i>instance</i> from the database associated with it. If <i>instance</i> has no associated database, delete-instance-records raises an error.
See also	update-records update-records-from-instance

## delete-records

Summary	Deletes rows fro	om a database table.
Package	sql	
Signature	delete-records	&key from where database =>
Arguments	from	A database table.
	where	An SQL conditional statement.
	database	A database.
Values	None.	
Description	ified by <i>from</i> in ment <i>database</i> sp	cords function deletes rows from a table spec- which the <i>where</i> condition is true. The argu- pecifies a database from which the records are and defaults to *default-database*.
See also	insert-record	s
	update-record	S

# delete-sql-stream

Function

Summary	Deletes a stream mand or result t	from the broadcast stream for SQL com- raffic.
Package	sql	
Signature	delete-sql-stre	eam stream &key type database => old-stream
Arguments	stream	The broadcast stream.
	type	A keyword.
	database	A database.
Values	old-stream	The deleted stream.
Description	The function delete-sql-stream deletes a stream from the broadcast stream for SQL command or result traffic.	
	• •	pe is one of :command or :result, and deter- a stream for SQL command or result traffic is
		ne of <i>type</i> is :command. The default value for mult-database*.
See also	add-sql-stream list-sql-stream sql-recording- sql-stream start-sql-recor stop-sql-recor	ms P ording

# disable-sql-reader-syntax

Function

Summary Turns off square bracket syntax.

Package sql

Signature	disable-sql-reader-syntax =>
Arguments	None.
Values	None.
Description	The function disable-sql-reader-syntax turns off square bracket syntax and sets state so that restore-sql-reader- syntax-state will make the syntax disabled if it is conse- quently enabled.
See also	enable-sql-reader-syntax locally-disable-sql-reader-syntax locally-enable-sql-reader-syntax restore-sql-reader-syntax-state

# disconnect

Summary	Closes a connection to a database.
Package	sql
Signature	disconnect &key <i>database</i> => nil
Arguments	database A database.
Values	nil
Description	The function disconnect closes a connection to a database specified by <i>database</i> and resets *default-database* if the database is successfully disconnected and only one other connection exists.
Example	(disconnect :database "test")

This chapter is applicable to the Enterprise version only

See also connect connected-databases database-name find-database status

# do-query

Macro

Summary	- 0	ds a variable to the results of a query, and exe- code using the bound variable.
Package	sql	
Signature	do-query &rest	args query &key database &rest body =>
Arguments	args	A set of bindings.
	query	A database query.
	database	A database.
	body	A Lisp code body.
Values	None.	
Description	ing of <i>args</i> on th	Tuery repeatedly executes <i>body</i> within a bind- ne attributes of each record resulting from n value of do-query is determined by the ng <i>body</i> .
	The default value	ue of <i>database</i> is *default-database*.
Example	entry in ename f	ode repeatedly binds the result of selecting an From the table emp to the variable name, and susing the Lisp function print.
		me) [select [ename] :from [emp]]) t name))

See also	loop
	map-query
	query
	select

# drop-index

Summary	Deletes an index from a database.	
Package	sql	
Signature	drop-index index &key database =>	
Arguments	indexThe name of an index.databaseA database.	
Values	None.	
Description	The function drop-index deletes <i>index</i> from <i>database</i> . The default value of <i>database</i> is *default-database*.	
See also	create-index drop-table	
drop-table	Functio	on
Summary	Deletes a table from a database.	
Package	sql	
Signature	drop-table table &key database =>	
Arguments	tableThe name of a table.databaseA database.	

Values	None.
Description	The function drop-table deletes table from a database.
	The default value of <i>database</i> is <b>*default-database</b> *.
See also	create-table
drop-view	Function
Summary	Deletes a view from a database.
Package	sql
Signature	drop-view view &key database =>
Arguments	view A view.
	database A database.
Values	None.
Description	The function drop-view deletes view from database.
	The default value of <i>database</i> is <b>*default-database</b> *.
See also	create-view
	drop-index drop-table

# drop-view-from-class

Function

Summary Deletes a view from a database based on a class defining the view.

Package sq1

Signature	drop-view-from	a-class <i>class</i> &key <i>database</i> =>
Arguments	class database	A class. A database.
Values	None.	
Description	table from data	cop-view-from-class deletes a view or base base based on <i>class</i> which defines that view. database has a default value of *default-data-
See also	create-view-f drop-view	rom-class

# enable-sql-reader-syntax

Summary	Turns on square bracket SQL syntax.
Package	sql
Signature	enable-sql-reader-syntax =>
Arguments	None.
Values	None.
Description	The function enable-sql-reader-syntax turns on square
·	bracket syntax and sets the state so that restore-sql- reader-syntax-state will make the syntax enabled if it is consequently disabled.

#### execute-command

#### Summary Executes an SQL expression. Package sql Signature execute-command sql-exp &key database => sql-exp Arguments Any SQL statement other than a query. database A database. Values None. Description The function execute-command executes the SQL command specified by *sql-exp* for the database specified by *database*, which has a default value of \*default-database\*. The argument *sql-exp* may be any SQL statement other than a query. See also query

### find-database

#### Function

Summary	Returns a databa	ase, given a database or database name.
Package	sql	
Signature	find-database (	database &optional errorp => database
Arguments	database errorp	A database. A boolean. Default value: t.
Values	database	A database.
Description		nd-database, given a database or the name of rns that database. If no database is found and

*errorp* is t, then find-database signals an error, otherwise it returns nil.

See also connect connected-databases database-name disconnect status

# initialize-database-type

Summary	Initializes a data	abase type.
Package	sql	
Signature	initialize-dat	abase-type &key database-type => type
Arguments	database-type	A database type.
Values	type	A database type.
Description	base type by loa according to the base-type* is n	itialize-database-type initializes a data- ading code and appropriate database libraries e value of <i>database-type</i> . If *default-data- not initialized, this function initializes it. It appe to the list of initialized types. The initial- advectory period in the list of initialized types. The initial- advectory period is returned.
Example	database-type (require "odbc (in-package sq (setf *default (initialize-da	1) -database-type* :odbc)

The ODBC database type is now initialized, and connections can be made to ODBC databases.

See also database-name \*initialized-database-types\* \*default-database-type\*

### \*initialized-database-types\*

Summary	A list of initialized database types.
Package	sql
Initial Value	nil
Description	The variable <b>*initialized-database-types*</b> contains a list of database types that have been initialized by calls to <b>initialize-database-types</b> .
See also	initialize-database-type

### insert-records

#### Function

Variable

Summary	Inserts a set of values into a table.	
Package	sql	
Syntax	insert-records database =>	&key into attributes values av-pairs query
Arguments	into	A database table.
	values	A list of values or a query expression.
	attributes	A list of attributes.
	av-pairs	A list of two-element lists.

	database A database.
Values	None.
Description	The function insert-records inserts values for attributes (or <i>av-pairs</i> ) into the table <i>into</i> . The argument values may be a list of values or a query expression. If attributes is supplied then values must be a corresponding list of values for each of the listed attribute names.
	The default value of <i>database</i> is <b>*default-database</b> *.
Example	In the first example, the LispWorks expression
	(insert-records :into [person] :values `("abc" "Joe" "Bloggs" 10000 3000 nil "plumber"))
	is equivalent to the following SQL:
	INSERT INTO PERSON VALUES ('abc','Joe', 'Bloggs',10000,3000,NULL,'plumber')
	In the second example, the LispWorks expression
	(insert-records :into [person] :attributes '(person_id income surname occupation) :values '("aaa" 10 "jim" "plumb"))
	is equivalent to the following SQL:
	INSERT INTO PERSON (PERSON_ID,INCOME,SURNAME,OCCUPATION) VALUES ('aaa',10,'jim','plumb')
	The following example demonstrates how to use <code>:av-pairs</code> .
	(insert-records :into [person] :av-pairs '((person_id "bbb") (surname "Jones")))
See also	delete-records update-records

# list-attributes

### Function

Summary	Returns a lis	t of attributes from a table in a database.
Package	sql	
Signature	list-attrib	utes table &key database => result
Arguments	table database	A table in the database. A database.
Values	result	A list of attributes.
Description		list-attributes returns a list of attributes <i>database</i> , which has a default value of *default-
See also	attribute-t list-tables	

## list-classes

Summary	Returns a list of	view classes connected to a given database.
Package	sql	
Signature	list-classes &	key database root-class test => result-list
Arguments	database	A database.
	root-class	A class.
	test	A test function.
Values	result-list	A list of class objects.

Description	The list-classes function collects all the classes below <i>root-class</i> (which defaults to 'standard-db-object) that are connected to the given database specified by <i>database</i> , and which satisfy the <i>test</i> function. The default for the <i>test</i> argument is 'identity.
	By default, list-classes returns a list of all the classes con- nected to the default database, *default-database*.

## list-sql-streams

Summary	Returns the individual streams recording SQL command or result traffic.		
Package	sql		
Signature	list-sql-streams &key type database => streams		
Arguments	type	A keyword.	
	database	A database.	
Values	streams	A list of streams.	
Description	The function list-sql-streams returns the individual streams recording SQL command or result traffic, since there may be multiple streams wrapped up into a single broadcast stream. The streams are returned as a list.		
	• •	pe is one of :command or :result, and deter- to return a list of streams for SQL command	
		ue of <i>type</i> is :command. The default value for ault-database <sup>*</sup> .	
See also	add-sql-stream delete-sql-st		

sql-recording-p sql-stream start-sql-recording stop-sql-recording

### list-tables

#### Function

Summary	Returns a list of	the table names in a database.
Package	sql	
Signature	list-tables &ke	y database => table-list
Arguments	database	A database.
Values	table-list	A list of table names.
Description		st-tables returns the list of table names in has a default value of *default-database*.
See also	create-table drop-table list-attribute	s

### locally-disable-sql-reader-syntax

#### Function

SummaryTurns off square bracket syntax and does not change syntax<br/>state.PackagesqlSignaturelocally-disable-sql-reader-syntax =>ArgumentsNone.

Values	None.
Description	The function locally-disable-sql-reader-syntax turns off square bracket syntax and does not change syntax state. This ensures that restore-sql-reader-syntax-state restores the current enable/disable state.
Example	The intended use of locally-disable-sql-reader-syntax is in a file:
	<pre>#.(locally-disable-sql-reader-syntax)</pre>
	<lisp []="" code="" not="" syntax="" using=""></lisp>
	<pre>#.(restore-sql-reader-syntax-state)</pre>
See also	disable-sql-reader-syntax
	enable-sql-reader-syntax locally-enable-sql-reader-syntax
	restore-sql-reader-syntax-state
	Terrer 21 Touget Diver Dence

# locally-enable-sql-reader-syntax

Summary	Turns on square bracket syntax and does not change syntax state.
Package	sql
Signature	locally-enable-sql-reader-syntax =>
Arguments	None.
Values	None.
Description	The function locally-enable-sql-reader-syntax turns on square bracket syntax and does not change the syntax state. This ensures that restore-sql-reader-syntax-state restores the current enable/disable state.

Example	The intended use of locally-enable-sql-reader-syntax is in a file:
	<pre>#.(locally-enable-sql-reader-syntax)</pre>
	<sql []="" commands="" syntax="" using=""></sql>
	#.(restore-sql-reader-syntax-state)
See also	disable-sql-reader-syntax enable-sql-reader-syntax locally-disable-sql-reader-syntax restore-sql-reader-syntax-state

# loop

Macro

Summary	Extends 100p to results.	provide a clause for iterating over query
Package	cl	
Signature		ar [type-spec] being {the each} 1} {in of} query-expression => result
Arguments	var	A variable.
	query-expression	An SQL query statement.
Values	result	A loop return value.
Description	The Common Lisp 100p macro has been extended with a clause for iterating over query results. This extension is available only when Common SQL has been loaded. For a full description of the rest of the Common Lisp 100p facility, including the possible return values, see Steele, <i>Common Lisp: the Language</i> , second edition.	
		The loop assigns the next record of the table <i>ar</i> . The record is represented in Lisp as a list.

Destructuring can be used in *var* to bind variables to specific attributes of the records resulting from *query-expression*. In conjunction with the panoply of existing clauses available from the loop macro, the new iteration clause provides an integrated report generation facility.

Example This extended loop example performs the following on each record returned as a result of a query: bind name to the query result, find the salary (if any) from an associated hash-table, increment a count for salaries greater than 20000, accumulate the salary, and print the details. Finally, it prints the average salary.

(loop for (name) being each record in [select [ename] :from [emp]] as salary = (gethash name \*salary-table\*) initially (format t "~&~20A~10D" 'name 'salary) when (and salary (> salary 20000)) count salary into salaries and sum salary into total and do (format t "~&~20A~10D" name salary) else do (format t "~&~20A~10A" name "N/A") finally (format t "~2&Av Salary: ~10D" (/ total salaries)))

See also do-query map-query query select

map-query

Function

Summary Returns the results of mapping a function across an SQL query statement.

Package sql

Signature	map-query outpu result	nt-type-spec function query-exp &key database =>
Arguments	output-type-spec	The output type specification.
	result-type	The result sequence type.
	function	A function.
	query-exp	An SQL query.
	database	A database.
Values	result	A sequence of type <i>output-type-spec</i> contain- ing the results of the map function.
Description	<i>tion</i> across the r	p-query returns the result of mapping <i>func-</i> esults of <i>query-expression</i> . The <i>output-type-spec</i> fies the type of the result sequence as per the map function.
	The default value	ue of <i>database</i> is <b>*default-database*</b> .
Example	This example bi and prints it.	inds name to each name in the employee table
		name) (print name)) ame] :from [emp] :flatp t])
See also	do-query loop print-query query select	

# print-query

Function

Summary Prints a tabulated version of records resulting from a query.

Package	sql	
Signature	print-query <b>qu</b> =>	ery-exp &key titles formats sizes stream database
Arguments	query-exp	An SQL query expression.
	titles	A list of strings.
	formats	A list of strings.
	sizes	A list.
	stream	An output stream.
	database	A database.
Values	None.	
Description	The print-query function takes a symbolic SQL query expression and formatting information and prints onto <i>stream</i> a table containing the results of the query. A list of strings to use as column headings is given by <i>tit</i> which has a default value of nil.	
	The <i>formats</i> argument is a list of format strings used to print each attribute, and has a default value of t, which means that ~A or ~VA are used if sizes are provided or computed.	
	The field sizes are given by <i>sizes</i> . It has a default which specifies that minimum sizes are compute	
	-	eam is given by <i>stream</i> , which has a default specifies that <b>*standard-output</b> * is used.
Examples	The following of salaries:	call prints out two even columns of names and
		<pre>select [surname] [income] :from [person]] titles '("NAME" "SALARY"))</pre>

This chapter is applicable to the Enterprise version only

See also map-query print-query select

#### query

### Function

Summary	Queries a datab	ase and returns a list of values.
Package	sql	
Signature	query sql-exp &k	ey database => result-list
Arguments	sql-exp database	An SQL query statement to be performed. A database.
Values	result-list	A list of values.
Description	the database spe ment given by q	ery is the basic SQL query function. It queries ecified by <i>database</i> with an SQL query state- <i>query-expression</i> and returns a list of values as argument <i>database</i> defaults to *default-
See also	do-query execute-commar loop map-query select	nd

### restore-sql-reader-syntax-state

Function

Summary Sets the enable/disable square bracket syntax state to reflect the last call to either disable-sql-reader-syntax Or enablesql-reader-syntax.

Package	sql
Signature	restore-sql-reader-syntax-state =>
Arguments	None.
Values	None.
Description	The function restore-sql-reader-syntax-state sets the enable/disable state of the square bracket syntax to reflect the last call to either enable-sql-reader-syntax Or disable- sql-reader-syntax. The default state of the square bracket syntax is disabled.
See also	disable-sql-reader-syntax enable-sql-reader-syntax locally-disable-sql-reader-syntax locally-enable-sql-reader-syntax

# rollback

Summary	Rolls back changes made to a database since the last commit.
Package	sql
Signature	rollback &key <i>database</i> => nil
Arguments	database A database.
Values	nil
Description	The function rollback rolls back changes made in <i>database</i> since the last commit, that is, changes made since the last commit are not recorded. The argument <i>database</i> defaults to *default-database*.

See also	commit with-transac	tion
select		Function
Summary	Selects data fro straints.	om a database given a number of specified con-
Package	sql	
Signature		selections &key all flatp set-operation distinct from having order-by database => result-list
Arguments	selections	A set of database identifiers or strings.
	all	A boolean.
	flatp	A boolean.
	set-operation	An SQL operation.
	distinct	A boolean.
	from	An SQL table.
	where	An SQL condition.
	group-by	An SQL condition.
	having	An SQL condition.
	order-by	An SQL condition.
	database	A database.
Values	result-list	A list of selections.
Description	The function	al art solocts data from database which has a

Description The function select selects data from *database*, which has a default value of \*default-database\*, given the constraints specified by the rest of the arguments. It returns a list of objects as specified by *selections*. By default, the objects will each be represented as lists of attribute values. The argument

*selections* consists either of database-identifiers or literal strings.

The *flatp* argument, which has a default value of nil, specifies if full bracketed results should be returned for each matched entry. If *flatp* is nil, the results are returned as a list of lists. If *flatp* is t, the results are returned as elements of a list, only if there is only one result per row. See the examples section for an example of the use of *flatp*.

The arguments *all, set-operation, distinct, from, where, group-by, having* and *order-by* have the same function as the equivalent SQL expression.

The select function is common across both the functional and object-oriented SQL interfaces. If *selections* refers to view classes then the select operation becomes object-oriented. This means that select returns a list of view-class instances, and slot-value becomes a valid SQL operator for use within the *where* clause.

SQL expressions used in the select function are specified using the square bracket syntax, once this syntax has been enabled using enable-sql-reader-syntax.

Examples The following is a potential query and result: (select [person\_id] [surname] :from [person]) => ((111 "Brown") (112 "Jones") (113 "Smith")) In the next example, the flatp argument is set to t, and the result is a simple list of surname values: (select [surname] :from [person] :flatp t) => ("Brown" "Jones" "Smith") See also print-query

## sql

### Function

Summary	Generates SQL from a set of expressions.		
Package	sql		
Signature	sql &rest args => sql-expression		
Arguments	args A set of expressions.		
Values	sql-expression An SQL expression.		
Description	The function $sql$ generates SQL from a set of expression given by <i>args</i> . Each argument to $sql$ is translated into SQ and then the <i>args</i> are concatenated with a single space between each pair. The rules for translation into SQL, ba on the type of each individual argument $x$ , are as follow		
	<pre>string =&gt; (format nil "'~A'" x)</pre>		
	nil => "NULL"		
	<pre>symbol =&gt; (symbol-name x)</pre>		
	<pre>number =&gt; (princ-to-string x)</pre>		
	list => (format nil "(~{~A~^,~})" (mapcar #'sql x))		
	<pre>vector =&gt; (format nil "~{~A~^,~}" (map 'list #'sql x))</pre>		
	sql-expression => x		
	Any other symbol => error		
See also	sql-expression sql-operation		

sql-operator

### sql-expression

### Function

Summary Generates an SQL expression from the given keywords.

The SQL Package

Package	sql		
Signature	sql-expression => <i>sql-result</i>	&key string table alias attribute type	
Arguments	string	A string.	
	table	A table in a database.	
	alias	A table alias.	
	attribute	An attribute.	
	type	A type.	
Values	sql-result	An SQL expression.	
Description	The function sq from the given l	1-expression generates an SQL expression keywords.	
	Valid combinations of the arguments are:		
	• string		
	• table		
	• <i>table</i> and <i>alias</i>		
	• table and attribute		
	• <i>table, attribute,</i> and <i>type</i>		
	• <i>table</i> or <i>alias,</i> and <i>attribute</i>		
	• <i>table</i> or <i>alias,</i> and <i>attribute</i> and <i>type</i>		
	• attribute		
	• <i>attribute</i> and	type	
See also	sql		
	sql-operation		
	sql-operator		

# sql-operation

Function

Summary	Generates an SQL statement from an operator and arguments.		
Package	sql		
Signature	sql-operation of	sql-operation op &rest args => sql-result	
Arguments	op	An operator.	
.,,,	args	A set of arguments for <i>op</i> .	
Values	sql-result	An SQL expression.	
Description	The function sql-operation takes an operator and its arguments, and returns an SQL statement. It is shorthand for (apply (sql-operator op) args).		
Example	The following code, uses sql-operation to produce an SQL expression.		
	(sql-expressi :from (list (sql- (sql- :where (sql-opera (sql-oper (sql-ex 3) (sql-oper	<pre>.on :table 'foo :attribute 'bar) .on :attribute 'baz) expression :table 'foo) expression :table 'quux)) tion 'or ration '&gt; pression :attribute 'baz) ration 'like pression :table 'foo :attribute 'bar)</pre>	
	The following SQL expression is produced.		

#<SQL-QUERY: "(SELECT FOO.BAR,BAZ FROM FOO,QUUX WHERE ((BAZ > 3) OR (FOO.BAR LIKE 'SU%')))"> See also sql sql-expression sql-operator

# sql-operator

### Function

Summary	Returns the SQ	L symbol for an operator.
Package	sql	
Signature	sql-operator s	ymbol => sql-symbol
Arguments	symbol	An SQL operator.
Values	sql-symbol	An SQL symbol.
Description		l-operator takes an operator as an argument SQL symbol for the operator.
See also	sql sql-expressio sql-operation	

# sql-recording-p

Summary	A predicate for determining if SQL command or result traffic is being recorded.	
Package	sql	
Signature	sql-recording-p &key type database => result-p	
Arguments	type	One of :command Or :result.
	database	A database.

Values	result-p	A boolean.	
Description		ו-recording-p returns ב if SQL command or being recorded, otherwise returns mil.	
	The keyword <i>type</i> can be one of <b>:command</b> or <b>:result</b> , and specifies whether to check if command or result traffic is being recorded.		
		ue of <i>type</i> is :command. The default value of ault-database*.	
See also	add-sql-stream delete-sql-stream list-sql-streams sql-stream start-sql-recording stop-sql-recording		

# sql-stream

Summary	Returns the bro mand or result	adcast stream used for recording SQL com- traffic
Package	sql	
Signature	sql-stream &key type database => stream	
Arguments	type database	One of :command or :result. A database.
Values	stream	A broadcast stream.
Description	The function sql-stream returns the broadcast stream used for recording SQL command or result traffic.	

The keyword *type* can be one of :command or :result, and specifies whether to return the broadcast stream for command or result traffic.

The default value of *type* is :command. The default value of *database* is \*default-database\*.

See also add-sql-stream delete-sql-stream list-sql-streams sql-recording-p start-sql-recording stop-sql-recording

### start-sql-recording

#### Function

Summary Starts recording SQL command or result traffic onto a broadcast stream.

Package sql

Signature start-sql-recording &key type database =>

Arguments *type* A keyword.

database A database.

Values None.

Description The function start-sql-recording starts recording SQL command or result traffic onto a broadcast stream. Initially the broadcast stream for commands or results is just \*standard-output\*.

The keyword *type* is one of :command or :result, and determines whether SQL command or result traffic is recorded.

The default value of *type* is :command. The default value for *database* is \*default-database\*.

See also add-sql-stream delete-sql-stream list-sql-streams sql-stream sql-recording-p stop-sql-recording

### status

Summary	Returns status information for the connected databases and initialized database types.	
Package	sql	
Signature	status &optional full =>	
Arguments	full A boolean.	
Values	None.	
Description	The function status prints status information to the standard output, for the connected databases and initialized database types. The argument <i>full</i> is nil by default, but if it is set to t, more detailed status information is returned.	
See also	connect connected-databases database-name disconnect find-database	

# stop-sql-recording

### Function

Summary	Stops recording SQL command or result traffic.		
Package	sql		
Signature	stop-sql-recording &key type database =>		
Arguments	type database	One of :command or :result. A database.	
Values	None.		
Description	The function stop-sql-recording stops recording SQL com- mand or result traffic.		
	The keyword <i>type</i> is one of <b>:command</b> or <b>:result</b> , and determines if SQL command or result traffic is no longer to be recorded.		
		ue of <i>type</i> is :command. The default value for ault-database <sup>*</sup> .	
See also	add-sql-stream delete-sql-stream list-sql-streams sql-recording-p sql-stream start-sql-recording		

## update-records

#### Function

Summary Changes the values of fields in a table.

Package sql

Signature	update-records database =>	table &key attributes values av-pairs where
Arguments	table	A database table.
	attributes	A set of columns.
	values	A set of values.
	av-pairs	An association list alternative to <i>attributes</i> and <i>values.</i>
	where	A condition.
	database	A database.
Values	None.	
Description	fields in <i>table</i> w	date-records changes the values of existing ith columns specified by <i>attributes</i> and <i>values</i> ere the <i>where</i> condition is true.
See also	delete-instand delete-record insert-record update-record	s

# update-records-from-instance

Summary	Updates a set o	f specified records in a database.
Package	sql	
Signature	update-records	-from-instance <i>instance</i> &key <i>database</i> =>
Arguments	instance database	An instance of a view class. A database.
Values	None.	

neric function
ed by <i>instance</i> . If the
abase, that database
ce is not yet associ-
for <i>instance</i> in the
ance becomes associ-

See also def-view-class delete-instance-records update-records

#### update-record-from-slot

**Generic Function** 

Summary	Updates an ind	ividual data item from a slot.
Package	sql	
Signature	update-record-	<pre>from-slot instance slot &amp;key database =&gt;</pre>
Arguments	instance	An instance of a view class.
	slot	A slot.
	database	A database.
Values	None.	
Description	individual data database is only database, in wh slot is initialized	ction update-record-from-slot updates an item in the column represented by <i>slot</i> . The used if <i>instance</i> is not yet associated with any ich case a record is created in <i>database</i> . Only l in this case; other columns in the underlying e default values. The argument <i>slot</i> is the

derived from the view class definition.

CLOS slot name; the corresponding column names are

See also def-view-class update-records-from-instance update-record-from-slots

## update-record-from-slots

**Generic Function** 

Summary	Updates an indi	ividual data item from a list of slots.
Package	sql	
Signature	update-record-:	from-slots <i>instance slots</i> &key <i>database</i> =>
Arguments	instance slots database	An instance of a view class. A list of slots. A database.
Values	None.	
Description	The generic function update-record-from-slots updates data in the columns represented by <i>slots</i> . The <i>database</i> is only used if <i>instance</i> is not yet associated with any database, in which case a record is created in <i>database</i> . Only <i>slots</i> are ini- tialized in this case; other columns in the underlying data- base receive default values. The argument <i>slots</i> contains the CLOS slot names; the corresponding column names are derived from the view class definition.	
See also	def-view-class update-records update-record-	s-from-instance

## with-transaction

Macro

Summary Performs a body of code within a transaction for a database.

Package	sql	
Signature	with-transaction &key database &body body => result	
Arguments	database A database type.	
	<i>body</i> A set of Lisp expressions.	
Values	<i>result</i> The value returned by <i>body</i> .	
Description	The macro with-transaction performs body within a trans- action for database (which defaults to *default-database*). The transaction is committed if the body finishes success- fully, otherwise the database is rolled back.	
Example	The following example shows how to use with-transaction to insert a new record, updates the department number of employes from 40 to 50, and removes employees whose sal- ary is higher than 300,000. If an error occurs anywhere in the body and an abort or throw is executed, none of the updates are committed.	
	<pre>(with-transaction (insert-record :into [emp] :attributes `(x y z) :values `(a b c)) (update-records [emp] :attributes [dept] :values 50 :where [= [dept] 40]) (delete-records :from [emp] :where [&gt; [salary] 300000]))</pre>	
See also	commit rollback	

# The STREAM package

This chapter describes the symbols available in the stream package that provide users with the functionality to define their own streams for use by the standard I/O functions.

Summary	A stream class for binary input.
Package	stream
Superclasses	fundamental-binary-stream fundamental-input-stream
Subclasses	None.
Description	The class fundamental-binary-input-stream provides a class for generating customized binary input stream classes. A method for stream-read-byte should be provided when using this class.

See also fundamental-binary-stream fundamental-input-stream stream-read-byte

# fundamental-binary-output-stream

Class

Summary	A stream class for binary output.
Package	stream
Superclasses	fundamental-binary-stream fundamental-output-stream
Description	The class fundamental-binary-output-stream provides a class for generating customized binary output stream classes. A method for stream-write-byte should be provided.
See also	fundamental-binary-stream fundamental-output-stream stream-write-byte

## fundamental-binary-stream

Summary	A class for binary streams.
Package	stream
Superclasses	fundamental-stream
Subclasses	fundamental-binary-input-stream fundamental-binary-output-stream
Description	The class fundamental-binary-stream is the superclass of the binary input and output stream classes. A method for stream-element-type should be provided for instantiable subclasses of this class.

#### See also fundamental-binary-input-stream fundamental-binary-output-stream fundamental-stream stream-element-type

## fundamental-character-input-stream

Summary	A class that should be included in stream classes for charac- ter input.
Package	stream
Superclasses	fundamental-character-stream fundamental-input-stream
Subclasses	None.
Description	The class fundamental-character-input-stream provides default methods for generic functions used for character input, and should therefore be included by stream classes concerned with character input. The user can provide meth- ods for these generic functions specialized on the user- defined class. Methods for other generic functions must be provided by the user.
See also	<pre>fundamental-character-stream fundamental-input-stream stream-clear-input stream-listen stream-peek-char stream-read-char stream-read-char-no-hang stream-read-line stream-read-line stream-read-sequence stream-unread-char</pre>

## fundamental-character-output-stream

Summary	A class that should be included in stream classes for charac- ter output.
Package	stream
Superclasses	fundamental-character-stream fundamental-output-stream
Subclasses	None.
Description	The class fundamental-character-output-stream provides default methods for generic functions used for character out- put, and should therefore be included by stream classes con- cerned with character output. The user can provide methods for these generic functions specialized on the user-defined class. Methods for other generic functions must be provided by the user.
See also	<pre>fundamental-character-stream fundamental-input-stream stream-clear-output stream-finish-output stream-force-output stream-start-line-p stream-terpri stream-line-column stream-write-char stream-write-sequence stream-write-string</pre>

## fundamental-character-stream

Class

Class

Summary A class whose inclusion provides a method for streamelement-type that returns CHARACTER.

Class

325

Package	stream
Superclasses	fundamental-stream
Subclasses	fundamental-character-input-stream fundamental-character-output-stream
Description	The class fundamental-character-stream is a superclass for character streams. Its inclusion provides a method for the generic function stream-element-type that returns the sym- bol CHARACTER.
See also	fundamental-character-input-stream fundamental-character-output-stream fundamental-stream stream-element-type

# fundamental-input-stream

Summary	A class whose inclusion causes input-stream-p to return t.
Package	stream
Superclasses	fundamental-stream
Subclasses	fundamental-binary-input-stream fundamental-character-input-stream
Description	The fundamental-input-stream class is a superclass to the binary and character input classes. Its inclusion causes the generic function input-stream-p to return t.
See also	fundamental-binary-input-stream fundamental-character-input-stream fundamental-stream input-stream-p

# fundamental-output-stream

Summary	A class whose inclusion causes $output-stream-p$ to return t.
Package	stream
Superclasses	fundamental-stream
Subclasses	fundamental-binary-output-stream fundamental-character-output-stream
Description	The fundamental-output-stream class is a superclass to the binary and character output classes. Its inclusion causes the generic function output-stream-p to return t.
See also	fundamental-binary-output-stream fundamental-character-output-stream fundamental-stream input-stream-p

## fundamental-stream

#### Class

Summary	A class whose inclusion causes streamp to return t.
Package	stream
Superclasses	standard-object stream
Subclasses	fundamental-binary-stream fundamental-character-stream fundamental-input-stream fundamental-output-stream
Description	The class fundamental-stream is a superclass to the funda- mental input, output, character and binary streams. Its inclu- sion causes streamp to return t.

See also close fundamental-binary-stream fundamental-character-stream fundamental-input-stream fundamental-output-stream open-stream-p

## stream-advance-to-column

Summary	Writes the required number of blank spaces to ensure that the next character will be written in a given column.		
Package	stream		
Signature	<pre>stream-advance-to-column stream column =&gt; result</pre>		
Arguments	stream	A stream.	
	column	An integer.	
Values	result	A boolean.	
Description	The generic function stream-advance-to-column writes enough blank spaces to <i>stream</i> to ensure that the next charac- ter is written at <i>column</i> . The generic function returns t if the operation is successful, or nil if it is not supported for this stream.		
	This function is intended for use by print and format ~t The default method uses stream-line-column and repeat calls to stream-write-char with a #\space character, and returns nil if stream-line-char returns nil.		
See also	stream-line-co	olumn	

# stream-clear-input

Generic Function

Summary	Implements clear-input.	
Package	stream	
Signature	<pre>stream-clear-input stream =&gt; nil</pre>	
Arguments	stream A stream.	
Values	nil	
Description	The generic function stream-clear-input implements clear-input. The default method is defined on fundamental-input-stream and does nothing.	
See also	fundamental-input-stream	

# stream-clear-output

Summary	Implements clear-output.
Package	stream
Signature	stream-clear-output <i>stream</i> => nil
Arguments	stream A stream.
Values	nil
Description	The generic function stream-clear-output implements clear-output. The default method is on fundamental-output-stream and does nothing.
See also	fundamental-output-stream

# stream-finish-output

## Generic Function

Summary	Implements finish-output.
Package	stream
Signature	<pre>stream-finish-output stream =&gt; nil</pre>
Arguments	stream A stream.
Values	nil
Description	The generic function stream-finish-output implements finish-output. The default method is on fundamental- output-stream and does nothing.
See also	fundamental-output-stream

# stream-force-output

Summary	Implements force-output.	
Package	stream	
Signature	<pre>stream-force-output stream =&gt; nil</pre>	
Arguments	stream A stream.	
Values	nil	
Description	The generic function stream-force-output implements force-output. The default method is on fundamental- output-stream and does nothing.	
See also	fundamental-output-stream	

## stream-fresh-line

## Generic Function

Summary	Used by fresh-	-line to start a new line on a given stream.
Package	stream	
Signature	stream-fresh-l	ine stream => bool
Arguments	stream	A stream.
Values	bool	A generalized boolean.
Description	The generic function stream-fresh-line is used by fresh- line to start a new line on a stream. The default method uses stream-start-line-p and stream-terpri. The result value is t if a new line is output successfully.	
See also	stream-start- stream-terpri	line-p

## stream-line-column

Summary	Returns the colu written.	umn number where the next character will be
Package	stream	
Signature	stream-line-co	lumn stream => column
Arguments	stream	A stream.
Values	column	An integer.
Description	umn number w	ction stream-line-column returns the col- here the next character will be written from this is not meaningful for the stream. This

function is used in the implementation of print and the format ~t directive. A method for this function must be defined for every character output stream class that is defined, although at its simplest it may be defined to always return nil.

See also fundamental-character-output-stream stream-start-line-p

#### stream-listen

**Generic Function** 

Summary	A function used available.	d by listen that returns t if there is input
Package	stream	
Signature	stream-listen	stream => bool
Arguments	stream	A stream.
Values	bool	A generalized boolean.
Description	The generic function stream-listen is used by listen and returns t if there is input available. The default method uses stream-read-char-no-hang and stream-unread-char. Most streams should define their own method as this is usually trivial and more efficient than the method provided.	
See also	stream-read-c stream-unread	-

#### stream-output-width

**Generic Function** 

Summary Used by the pretty printer to determine the output width when \*print-right-margin\* is nil.

Package	stream	
Signature	stream-output-	width stream => result
Arguments	stream	A stream.
Values	result	An integer or mil.
Description	pretty printer to right-margin* stream in units o	ction stream-output-width is used by the determine the output width when *print- is nil. It returns <i>result</i> , the integer width of of ems, or nil if the width is not known. The provided by fundamental-stream returns
See also	fundamental-st	cream

# stream-peek-char

Summary	-	on used by peek-char that returns a charac- ream without removing it from the stream
Package	stream	
Signature	stream-peek-cha	ar stream => result
Arguments	stream	A stream.
Values	result	A character or <b>:EOF</b> symbol.
Description	The generic function stream-peek-char is used to implement peek-char, and corresponds to a peek-type of nil. The default method reads a character from the stream without removing it from the stream buffer, by using stream-read- char and stream-unread-char.	

See also	stream-listen
	stream-read-char
	stream-unread-char

## stream-read-byte

#### Generic Function

Summary	A generic function used by read-byte to read an integer or :EOF symbol from a binary stream.	
Package	stream	
Signature	stream-read-by	te stream => result
Arguments	stream	A stream.
Values	result	An integer or :EOF symbol.
Description	byte, and return	ction stream-read-byte is used by read- ns either an integer or an :EOF symbol read stream specified by <i>stream</i> .
	When defining a on this generic f	a binary stream, a method should be defined Function.
See also	fundamental-b: fundamental-b:	inary-input-stream inary-stream

## stream-read-char

Generic Function

Summary Read one character from a stream.

Package stream

Signature stream-read-char stream => character

#### The STREAM package

Arguments	stream	A stream.
Values	character	A character or the <b>:EOF</b> symbol.
Description	The generic function stream-read-char reads one item from <i>stream</i> . The item read is either a character or the end of file symbol :EOF if the stream is at the end of a file. Every sub- class of fundamental-character-input-stream must define a method for this function.	
See also	fundamental-ch stream-unread-	naracter-input-stream -char

# stream-read-char-no-hang

Summary		character from the stream, an <b>:EOF</b> if the ached, or mil if no input is currently available.
Package	stream	
Signature	stream-read-ch	ar-no-hang stream => result
Arguments	stream	A stream.
Values	result	Either a character, an <b>:EOF</b> symbol, or nil.
Description	ments read-cha from the stream input is availab mental-charact char which is su	ction stream-read-char-no-hang imple- hr-no-hang. It returns either a character read , an :EOF if end-of-file is reached, or nil if no le. The default method provided by funda- ter-input-stream simply calls stream-read- ufficient for file streams, but interactive define their own method.
See also	fundamental-cl	naracter-input-stream

## stream-read-line

Summary	Returns a string	g read from a stream.
Package	stream	
Signature	stream-read-li	ne stream => result terminated
Arguments	stream	A stream.
Values	result terminated	A string or <b>: EOF</b> . A boolean.
Description	The generic function stream-read-line reads a line of char- acters from <i>stream</i> and returns this line as a string. If the string is terminated by an end-of-file instead of a newline then <i>terminated</i> is t.	
		thod uses repeated calls to stream-read-char, m-element-type to determine the element- t.
See also	fundamental-c stream-elemen stream-read-c	

# stream-read-sequence

Summary	Reads a numbe	r of items from a stream into a sequence.
Package	stream	
Signature	stream-read-se	<pre>quence stream sequence start end =&gt; index</pre>
Arguments	stream	A stream.
	sequence	A sequence.

	start	An integer.
	end	An integer.
Values	index	An integer.
Description	The generic function stream-read-sequence reads from <i>stream</i> into <i>sequence</i> . Elements from the <i>start</i> of <i>sequence</i> are replaced by elements from <i>stream</i> until <i>end</i> in <i>sequence</i> or the end-of-file in <i>stream</i> is reached. The index of the first element in <i>sequence</i> that is not replaced is returned.	
	input-stream char and uses A default meth input-stream byte and also u	od is provided by fundamental-character- which makes repeated calls to stream-read- (setf elt) to insert characters into sequence. od is provided by fundamental-binary- that makes repeated calls to stream-read- uses (setf elt) to insert bytes into sequence. may lead to error if the sequence is of inappro-
See also		-

# stream-start-line-p

Summary	A generic function that returns $\mathbf{t}$ if the stream is positioned at the beginning of a line.
Package	stream
Signature	<pre>stream-start-line-p stream =&gt; result</pre>
Arguments	stream A stream.

Values	result	A boolean.
Description	stream is positio	action stream-start-line-p returns t if oned at the beginning of a line, and nil other- issible to define a method that always returns
	also indicates the stream-start- and stream-linuusing variable-	ugh a value of 0 from stream-line-column ne beginning of a line, there are cases where line-p can be meaningfully implemented ne-column cannot. For example, for a window width characters the column number is not il, whereas the beginning of a line has a clear
	The default method for stream-line-start fundamental-character-output-stream us column. Therefore, if this is defined to return should be provided for either stream-start stream-fresh-line.	
See also	fundamental-c stream-fresh- stream-line-c	
stream-terpri		Generic Function

Summary	Writes an end o	f line to a stream.
Package	stream	
Signature	stream-terpri	stream => nil
Arguments	stream	A stream.
Values	nil	

#### The STREAM package

Description	The generic function stream-terpri writes an end of line to
	stream, as for terpri. The default method for stream-terpri
	is (stream-write-char <i>stream</i> #\newline).

See also stream-write-char

### stream-unread-char

Summary	Undoes the last call to stream-read-char.		
Package	stream		
Signature	stream-unread-	char stream character => n	il
Arguments	stream character	A stream. A character.	
Values	nil		
Description	call to stream-r	ction stream-unread-cha read-char, as in unread-c -character-input-strea function.	har. Every subclass
See also	fundamental-c	haracter-input-stream	
stream-write-b	oyte		Generic Function
Summary	A generic funct a binary stream	ion used by write-byte to	o write an integer to

- Package stream
- Signature stream-write-byte stream => result

Arguments	stream	A stream.
-----------	--------	-----------

Values *result* An integer.

Description The generic function stream-write-byte is used by writebyte, and writes an integer to the binary stream specified by *stream*. When defining a binary stream, a method should be defined

When defining a binary stream, a method should be defined on this generic function.

See also fundamental-binary-output-stream fundamental-binary-stream

## stream-write-char

Summary	Writes a charact	ter to a specified stream.
Package	stream	
Signature	stream-write-c	har stream character => character
Arguments	stream character	A stream. A character.
Values	character	A character.
Description	stream. Every su	ction stream-write-char writes <i>character</i> to lbclass of fundamental-character-output- ve a method defined for this function.
See also	fundamental-cl	naracter-output-stream

# stream-write-sequence

Summary	Writes a subsequence of a sequence to a stream.	
Package	stream	
Signature	stream-write-s	equence stream sequence start end => result
Arguments	stream	A stream.
	sequence	A sequence.
	start	An integer.
	end	An integer.
Values	result	A sequence.
Description	The generic function stream-write-sequence is used by write-sequence to write a subsequence of <i>sequence</i> delimited by <i>start</i> and <i>end</i> to <i>stream</i> .	
	output-stream and then uses s default method put-stream tha	od is provided by fundamental-character- that tests each element of <i>sequence</i> in turn, tream-write-char or produces an error. A is provided by fundamental-binary-out- t tests each element of <i>sequence</i> in turn, and um-write-byte or produces an error.
See also	fundamental-binary-output-stream fundamental-character-output-stream stream-read-sequence stream-write-byte stream-write-char	

# stream-write-string

Summary	Used by write- stream.	string to write a string to a character output
Package	stream	
Signature	stream-write-s result	tring stream string &optional start end =>
Arguments	stream	A stream.
	string	A string.
	start	An integer.
	end	An integer.
Values	result	A string.
Description	The generic function stream-write-string is used by write-string to write string to stream. The string can, optionally, be delimited by start and end.	
		thod provided by fundamental-character- uses repeated calls to stream-write-char.
See also	fundamental-cl stream-write-	haracter-output-stream char

The STREAM package

# The SYSTEM Package

This chapter describes symbols available in the **system** package.

## augmented-string

Summary	The augmented	string type.
Package	system	
Signature	augmented-strin	ng length
Arguments	length	The length of the string (or *, meaning any).
Description	This is the string to:	type that can hold any character. Equivalent
	(vector character	r length)

## augmented-string-p

Function

Type

Summary Tests if an object is an augmented string.

The SYSTEM Package

Package	system	
Signature	<pre>augmented-string-p object =&gt; bool</pre>	
Arguments	object	The object to be tested.
Values	bool	t if <i>object</i> is an augmented string; נות other- wise.
Description	This is the predicate for augmented strings.	
See also	augmented-string	

# default-eol-style

## Function

Summary	Provides a default end of line style for a file.	
Package	system	
Signature	default-eol-sty	yle pathname ef-spec buffer length => new-ef-spec
Arguments	pathname	Pathname identifying location of buffer.
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.
Values	new-ef-spec	A new external format spec created by merging <i>ef-spec</i> with the encoding that was found.
Description	Merge <i>ef-spec</i> with (:default :eol-style :crlf) on MS Windows, (:default :eol-style :lf) on UNIX. This is usually used as the last function on its list.	

## default-file-encoding

#### Function

Summary	Provides a default encoding in cases where an encoding can- not be deduced.	
Package	system	
Signature	default-file-en spec	ncoding pathname ef-spec buffer length => new-ef-
Arguments	pathname	Pathname identifying location of buffer.
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.
Values	new-ef-spec	Default external format spec created by merging <i>ef-spec</i> with the encoding that was found.
Description	Consults the ANSI code page in Windows, and merges <i>ef-spec</i> with :shift-jis for 932 and 10001, and :latin-1 otherwise. This is usually used as the last function on the *FILE-ENCODING-DETECTION-ALGORITHM* list.	
	Consults the locale on UNIX.	
See also	*file-encoding-detection-algorithm*	

## detect-encoding-in-file

Function

Summary Determines which type of encoding is used in a buffer.

The SYSTEM Package

Package	system	
Signature	detect-encoding ef-spec	g-in-file pathname ef-spec buffer length => new-
Arguments	pathname	Pathname identifying location of <i>buffer</i> .
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.
Values	new-ef-spec	A new external format spec created by merging <i>ef-spec</i> with the encoding that was found.
Description	Assume the encoding is one of :latin-1 or :unicode, and try to determine which of these it is, by looking for distinctive byte sequences in <i>buffer</i> up to <i>length</i> . If found, merge <i>ef-spec</i> with that encoding.	
See also	*file-encoding	g-detection-algorithm*

# detect-eol-style

#### Function

Summary	Detects the end	of line style of a file.
Package	system	
Signature	detect-eol-sty	le pathname ef-spec buffer length => new-ef-spec
Arguments	pathname	Pathname identifying location of buffer.
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.

	length	Length (an integer) up to which <i>buffer</i> should be examined.	
Values	new-ef-spec	A new external format spec created by merging <i>ef-spec</i> with the encoding that was found.	
Description	When the encoding in <i>ef-spec</i> has foreign type (unsigned- byte 8), search <i>buffer</i> up to <i>length</i> for the first occurrence of the byte (10). If found, and it is preceded in <i>buffer</i> by (13), merge <i>ef-spec</i> with		
	(:default :eol-style :crlf)		
	If found and is not preceded by (13), merge <i>ef-spec</i> with		
	(:default :eol-style :lf)		
	Thus a complete external format spec is constructed. Other wise, return <i>ef-spec</i> . When the encoding in <i>ef-spec</i> has foreign type (unsigned- byte 16), search <i>buffer</i> up to <i>length</i> for the first occurrence the byte sequence (13 0 10). If found, merge <i>ef-spec</i> with		
	(:default :eol-style :crlf)		
		not found, search <i>buffer</i> up to <i>length</i> for (10 0) und, merge <i>ef-spec</i> with	
	(:default :eol-style :lf)		
	Thus a complete external format spec is constructed. Other- wise, return <i>ef-spec</i> .		
See also	*file-eol-sty	le-detection-algorithm*	

# detect-japanese-encoding-in-file

Function

Summary Determines which type of Japanese encoding is used in a buffer.

The SYSTEM Package

Package	system	
Signature	<pre>detect-japanese-encoding-in-file pathname ef-spec buffer length =&gt; new-ef-spec</pre>	
Arguments	pathname	Pathname identifying location of buffer.
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.
Values	new-ef-spec	A new external format spec created by merging <i>ef-spec</i> with the Japanese encoding that was found.
Description	Assume the encoding is one of :jis, :sjis, :euc, :unicode and :ascii, and try to determine which of these it is, by look- ing for distinctive byte sequences in <i>buffer</i> up to <i>length</i> . If found, merge <i>ef-spec</i> with that encoding.	
See also	*file-encoding	g-detection-algorithm*

# \*extended-spaces\*

#### Variable

Summary	Extends the notion of space to include more than just the space character.
Package	system
Initial value	nil
Description	When this variable is true, the concept of "space" is extended from just #\space to include other appropriate characters. The default is nil, for ANS compliance, but we recommend that you set it to t.

This variable controls how the format directives ~:c and ~:@c output graphic characters which have an empty glyph. When this variable is t, all such characters are output using the name:

(format nil "~:C" #\No-break-space) -> "No-Break-Space" (format nil "~:C" (code-char #x3000)) -> "Ideographic-Space"

When false, only one such character is output using the name:

(format nil "~:C" #\Space) -> "Space"
(format nil "~:C" #\No-break-space) -> " "
(format nil "~:C" (code-char #x3000)) -> " "

It also affects whitespace-char-p.

See also whitespace-char-p

## \*file-encoding-detection-algorithm\*

#### Variable

Summary	List of functions to call to work out an encoding.
Package	system
Initial value	(FIND-FILENAME-PATTERN-ENCODING-MATCH FIND-ENCODING-OPTION DETECT-ENCODING-IN-FILE DEFAULT-FILE-ENCODING)
Description	Functions on this list take four arguments—the pathname of the file; an external format spec; a vector of element-type (unsigned-byte 8) which contains the first bytes of the file; and a non-negative integer which is the maximum extent of buffer to be searched. This length argument is 0 in the case that the file does not exist, or the direction is :output. They return an external format spec, which normally is either ef-

	spec unmodified, or the result of merging ef-spec with another external format spec via merge-ef-specs.
Example	If you want to inspect the attribute line and then fall back to a default if not found set the variable to the following:
	(FIND-ENCODING-OPTION DEFAULT-FILE-ENCODING)
See also	find-filename-pattern-encoding-match find-encoding-option detect-encoding-in-file detect-japanese-encoding-in-file default-file-encoding

## file-encoding-resolution-error

#### Condition

Summary	An error type to be deduced.	o signal when an external file format cannot
Package	system	
Superclasses	error	
Slots	pathname	Pathname identifying location of file con- taining unrecognizable format.
	external-format	An external file format.
	element-type	An element type.
Description	An error type signalled when open, load Or compile-file fail to detect an external format to use. The pathname slot contains the pathname as defaulted by the caller. The <i>external-format</i> and <i>element-type</i> slots contain the arguments the caller received.	
See also	guess-externa	l-format

## \*file-eol-style-detection-algorithm\*

Summary	List of functions for determining the end of line style of a file.
Package	system
Description	Functions on this list satisfy the same specifications as for those in <i>*file-encoding-detection-algorithm*</i> . However they will only be passed an external format spec with the name already determined.
Initial value	(detect-eol-style default-eol-style)
See also	detect-eol-style default-eol-style

## \*filename-pattern-encoding-matches\* Variable

Summary	An association of filename patterns to external format specs.
Package	system
Initial value	(("TAGS" . (:latin-1 :eol-style :lf)))
Description	An alist of filename patterns to external format specs.
See also	*file-encoding-detection-algorithm*

# find-encoding-option

#### Function

Variable

Summary	Examines a buffer for an encoding option.
Package	system
Signature	<pre>find-encoding-option pathname ef-spec buffer length =&gt; result</pre>

#### The SYSTEM Package

Arguments	pathname	Pathname identifying location of <i>buffer</i> .
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.
Values	result	The result of reading the value returned from the encoding Or external-format option as a Lisp expression in the keyword package.
Description	Looks in the file options (EMACS-style -*- line) for an option called encoding or external-format, with value value. If found, read value as a Lisp expression in the keyword package and return the result.	
See also	*file-encoding-detection-algorithm*	

# find-filename-pattern-encoding-match Function

Summary	Finds the encoding of a file based on the filename.	
Package	system	
Signature	<pre>find-filename-pattern-encoding-match pathname ef-spec buffer length =&gt; new-ef-spec</pre>	
Arguments	pathname	Pathname identifying location of <i>buffer</i> .
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.

Values	new-ef-spec	Corresponding external format spec for the encoding that was found.
Description	of *filename-p	name (using pathname-match-p) with elements attern-encoding-matches*. If a match is ne corresponding external format spec.
See also	*file-encoding	g-detection-algorithm*

# guess-external-format

Summary	Tries to work ou	ut the external format
Package	system	
Signature	guess-external	-format pathname ef-spec buffer length => ef-spec
Arguments	pathname	Pathname identifying location of buffer.
	ef-spec	An external format spec.
	buffer	A buffer whose contents are examined.
	length	Length (an integer) up to which <i>buffer</i> should be examined.
Values	ef-spec	An external format spec.
Description	detection-algorithm returned it is us next function. If returned by the of type file-en caller offers a re nal-format and functions on the	rn, functions on the list *file-encoding- orithm*. If a complete external format spec is ed, otherwise the return value is passed to the f the name of the external format spec last function on this list is :default, an error coding-resolution-error is signalled. The estart for trying again with respecified exter- d/or element-type arguments. Otherwise, e list *FILE-EOL-STYLE-DETECTION- called in turn. If a complete external format

	spec is returned it is used, otherwise the return value is passed to the next function. If the external format spec returned by the last function on this list does not contain :eol-style, an error of type file-encoding-resolution-
	error is signalled.
See also	<pre>*file-encoding-detection-algorithm* *file-eol-style-detection-algorithm* file-encoding-resolution-error</pre>

## in-static-area

## Macro

Summary	Allocates the obstatic area.	jects produced by the specified forms to the
Package	system	
Signature	in-static-area	&rest body => result
Arguments	body	The forms for which you want the garbage collector to allocate space in the static area.
Values	result	The result of executing <i>body</i> .
Description	static area. Obje	jects produced by the specified forms to the cts in the static area are not moved, though e collected when there is no longer a pointer
Example	(system:in-stat	tic-area (make-string 10))
See also	enlarge-static staticp	

# \*line-arguments-list\*

Variable

Summary	List of the command line arguments used when LispWorks was invoked.
Package	system
Initial Value	nil
Description	This variable contains a list of the arguments with which LispWorks was called. Its value is nil in the absence of any such arguments.

# mark-and-sweep

Summary	Garbage collects	s a specified generation.
Package	system	
Signature	mark-and-sweep	gen-number => bytes
Arguments	gen-number	0 for the most recent generation, 1 for the most recent two generations, and so on up to a maximum (usually 3). Numbers outside this range signal an error.
Values	bytes	The number of bytes allocated in that gener- ation.
Description	tion of storage ( tion forces the g generations. Thi ings of program tively, performa	a is used to garbage-collect a specified genera- and all lower generations). A call to this func- arbage collector to scan the specified is can be of use in obtaining consistent tim- s that require memory allocation. Alterna- nce can sometimes be improved by forcing a on, when it is known that little memory has

been allocated since a previous collection, rather than waiting for a later, more extensive collection. For example, the function could be called outside a loop that allocates a small amount of memory.

It is specially helpful to make and sweep generation 2 when large, long-lived data structures become garbage, because by default it is never marked and swept. The higher the generation number the more time the mark-and-sweep takes, but also the more space recovered.

Examples (mark-and-sweep 0) ; collect most recent generation (mark-and-sweep 3) ; collect all generations See also avoid-gc get-gc-parameters gc-if-needed normal-gc set-gc-parameters without-interrupts with-heavy-allocation

## merge-ef-specs

Summary	Creates a new e format specs.	external format spec from two other external
Package	system	
Signature	merge-ef-specs	ef-spec1 ef-spec2 => ef-spec
Arguments	ef-spec1 ef-spec2	The first external format spec to use. The second external format spec to use.
Values	ef-spec	The resultant external format spec created from information in <i>ef-spec1</i> and <i>ef-spec2</i> .

If *ef-spec1* is :default, return *ef-spec2*.

If ef-spec1 is a list beginning with :default and ef-spec2 is a
symbol, return (ef-spec2 . (cdr ef-spec1))

If *ef-spec1* is a list beginning with :default and *ef-spec2* is a list, return ((car *ef-spec2*) . *parameters*) where *parameters* is a destructuring lambda list containing the parameter descriptions of *ef-spec1* and any additional parameter descriptions from *ef-spec2*.

If *ef-spec1* is a symbol other than :default, or a list beginning with a symbol other than :default, and *ef-spec2* is a symbol, return *ef-spec1*.

If *ef-spec1* is a symbol other than :default, and *ef-spec2* is a list, return (*ef-spec1* . (cdr *ef-spec2*)).

If *ef-spec1* is a list beginning with a symbol other than :default, and *ef-spec2* is a list, return ((car *ef-spec1*) . *parameters*) where *parameters* is a destructuring lambda list containing the parameter descriptions of *ef-spec1* and any additional parameter descriptions from *ef-spec2*.

## object-address

Summary	Returns the address of the given <i>object</i> as an integer.	
Package	system	
Signature	object-address	s object => address
Arguments	object	The object whose address should be returned.

Values	address	The address of <i>object</i> . An integer.
Description	that the address	lress of the given <i>object</i> as an integer. Note is is likely to change during garbage collection hould be used for debugging purposes only.
Example		the address returned by sys:object- ame as the one printed by the print-object eric-function.
	address = 1cff	et ((gf #'initialize-instance)) (format t "address = ~X~%gf = ~S" (sys:object-address gf) gf)) 778 D-GENERIC-FUNCTION INITIALIZE-INSTANCE
See also	pointer-from-	address

# pointer-from-address

Summary	Returns the object into which the given address is pointing.	
Package	system	
Signature	pointer-from-a	ddress address => object
Arguments	address	The address of the object to return. An integer.
Values	object	The object pointed to by <i>address</i> .
Description	pointing. Note to object after a ga	ect into which the given integer <i>address</i> is that this address may not be pointing into this rbage collection, unless the object is static and d by another Lisp variable or object.

Example CL-USER 1 > (setq static-symbol (sys:in-static-area (make-symbol "THIS-SYMBOL"))) #:THIS-SYMBOL CL-USER 2 > (sys:object-address static-symbol) 25754040 CL-USER 3 > (sys:pointer-from-address 25754040) #:THIS-SYMBOL CL-USER 4 > (eq \* static-symbol) T CL-USER 5 > See also object-address

## \*sg-default-generation\*

Special Variable

Summary	Generation that new stack groups are allocated to by default.
Package	system
Initial Value	nil
Description	*sg-default-generation* is initially set to nil, thereby indi- cating that new stack groups should be allocated to the youngest generation, unless overwritten.

## \*sg-default-size\*

Variable

Summary Default initial size of a stack group, in 32 bit words.

Package system

Initial Value 16000

#### The SYSTEM Package

Description	This variable can be bound around a call to a process creation function. Note that setting the global value of this variable affects the size of all system processes too, so this is not rec- ommended.
Example	To create a process with a stack of 32000 words:
	<pre>(let ((lw:*sg-default-size* 32000))  (mp:process-run-function "Larger stack" '()     #'(lambda (print (lw::current-stack-length)))))</pre>

## simple-augmented-string

Summary	The simple augn	nented string type.
Package	system	
Signature	simple-augmente	d-string length
Arguments	length	The length of the string (or *, meaning any).
Description	-	e version of augmented-string, that is, the nple. Equivalent to: <i>character length</i> )
See also	augmented-stri	

## simple-augmented-string-p

#### Function

Type

g.
Ę

Package system

Signature simple-augmented-string-p object => bool

Arguments *object* The object to be tested.

Values	bool	t if <i>object</i> is a simple augmented string; nil otherwise.
Description	This is the p	predicate for simple augmented strings.
See also	simple-aug	mented-string

## \*stack-overflow-behaviour\*

## Variable

Summary	Controls behavio	or when stack overflow occurs.
Package	system	
Initial Value	:error	
Description	This variable con	ntrols behavior when stack overflow occurs.
	When <b>*stack-o</b> v Works signals ar	verflow-behaviour* is set to :error, Lisp- n error.
	automatically to	:warn, LispWorks increases the stack size accommodate the overflow, but prints a e to signal that this has happened.
	When it is set to silently.	<pre>:warn nil, LispWorks increases stack size</pre>
staticp		Function
Summary	Specifies whethe memory.	er a given object has been allocated in static
Package	system	
Signature	staticp obj => 1	bool
Arguments	obj	An object.

Values	bool	t if the object is allocated in static memory; הון otherwise.
Description	This predicate of the second s	can be used on an object to find out whether it tatic memory.
	eign language i ory. The Lisp re however. There even though the static memory.	ations made by Lisp — for example in a for- nterface program — are made in static mem- presentations of these alien objects are <i>not</i> , fore staticp applied to an alien returns nil e alien instance itself is really allocated in To establish this, you can check the pointer to ce within its Lisp representation (a structure).

# The WIN32 package (including DDE)

This chapter describes the symbols available in the win32 package, including reference entries for Dynamic Data Exchange (DDE). Symbols from this package can be used to allow your Common Lisp applications to exchange data using DDE. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

The functions are listed in three sections: general win32 symbols not concerned with DDE, the client interface reference entries which list all the DDE functions involved in providing client DDE functionality, and the server interface reference entries, which cover DDE server functionality.

## 18.1 General WIN32 symbols

## \*multibyte-code-page-ef\*

Variable

Summary Holds the external format corresponding to the current Windows multi-byte code page.

Package win32

- Description This variable holds the external format corresponding to the current Windows multi-byte code page. It is automatically initialized to the right value, when the image is started. If you change the code page (using \_setmbcp), you need to set this variable, too.
- See also character

## 18.2 DDE client interface reference entries

## dde-advise-start

Summary	Sets up an advise loop on a specified data item for a conver- sation.	
Package	win32	
Signature	dde-advise-start conversation item &key key function format datap type errorp => result	
Arguments	conversation	A conversation object.
	item	A string or symbol.
	key	An object.
	function	A function name.
	format	A clipboard format specifier.
	datap	A boolean.
	type	A keyword.
	errorp	A boolean.
Values	result	A boolean.
Description	The dde-advise-start function sets up an advise loop for the data item specified by <i>item</i> on the specified <i>conversation</i> .	

The argument format should be one of the following:

- A DDE format specifier, consisting of either a standard clipboard format or a registered clipboard format.
- A string containing either the name of a standard clipboard format (without the CF\_ prefix), or the name of a registered clipboard format.
- A symbol, in which case its print name is taken to specify the clipboard format.
- The keyword :text the default value of *format*. The keyword :text is treated specially. If supported by the server it uses the CF\_UNICODETEXT clipboard format, otherwise it used the CF\_TEXT format.

The argument *type* specifies how the response data should be converted to a Lisp object. For text formats, the default value indicates that a Lisp string should be created. The value <code>:string-list</code> may be specified to indicate that the return value should be taken as a tab-separated list of strings; in this case the Lisp return value is a list of strings. The default conversation class only supports text formats, unless *type* is specified as <code>:foreign</code>, which can be used with any clipboard format. It returns a <code>clipboard-item</code> structure, containing a foreign pointer to the data, the data length, and the format identifier.

If *datap* is t (the default value), a hot link is established, where the new data is supplied whenever it changes. If *datap* is nil, a warm link is established, where the data is not passed, and must be explicitly requested using dde-request.

The argument *key* is used to identify this link. If specified as nil (the default value), it defaults to the conversation. Multiple links are permitted on a conversation with the same *item* and *format* values, as long as their *key* values differ.

If the link is established, the return value *result* is t. If the link could not be established, the behavior depends on the value of *errorp*. If *errorp* is t (the default value), LispWorks signals

an error. If it is mil, the function returns mil to indicate failure.

If the link is established, the function *function* is called whenever the data changes. If *function* is nil (the default value), then the generic function dde-client-advise-data will be called.

The function specified by *function* should have a lambda list similar to the following:

key item data &key conversation &allow-other-keys

The arguments *key* and *item* identify the link. The argument *data* contains the new data for hot links; for warm links it is nil.

See also dde-advise-start\* dde-advise-stop dde-client-advise-data

## dde-advise-start\*

Summary	Sets up an advise loop for a specified data item for an auto- matically managed conversation.	
Package	win32	
Signature	dde-advise-start* service topic item &key key function format datap type errorp connect-error-p new-conversation-p => result	
Arguments	service	A string or symbol.
	topic	A string or symbol.
	item	A string or symbol.
	key	An object.
	function	A function name.

	format	A clipboard format specifier.	
	datap	A boolean.	
	type	A keyword.	
	errorp	A boolean.	
	connect-error-p	A boolean.	
	new-conversation-p		
		A boolean.	
Values	result	A boolean.	
Description	The dde-advise-start* function is similar to the dde- advise-start, and sets up an advise loop for the data item specified by <i>item</i> on a conversation recognizing the <i>ser-</i> <i>vice/topic</i> pair.		
	See dde-advise-start for information on the <i>format, type</i> , and <i>datap</i> arguments.		
	The argument <i>key</i> is used to identify this link. If specified as nil (the default value), it defaults to the conversation. Multiple links are permitted on a conversation with the same <i>item</i> and <i>format</i> values, as long as their <i>key</i> values differ.		
	If the link is established, the return value <i>result</i> is t. If the link could not be established, the behavior depends on the value of <i>errorp</i> . If <i>errorp</i> is t (the default value), LispWorks signals an error. If it is nil, the function returns nil to indicate failure.		
	If the link is established, the function <i>function</i> will be called whenever the data changes. If <i>function</i> is nil (the default value), the generic function dde-client-advise-data will be called.		
	The function sp similar to the fo	ecified by <i>function</i> should have a lambda list llowing:	
	key item data &k	ey conversation &allow-other-keys	

The arguments *key* and *item* identify the link. The argument *data* contains the new data for hot links; for warm links it is nil.

See also dde-advise-start dde-advise-stop dde-advise-stop\* dde-client-advise-data

## dde-advise-stop

Summary	Removes a link from a conversation specified by a given item and key.	
Package	win32	
Signature	dde-advise-stop conversation item &key key format errorp disconnectp no-advise-ok => result	
Arguments	conversation	A conversation object.
	item	A string or symbol.
	key	An object.
	format	A clipboard format specifier.
	errorp	A boolean.
	disconnectp	A boolean.
	no-advise-ok	A boolean.
Values	result	A boolean.
Description	The function dde-advise-stop removes a particular link from <i>conversation</i> specified by <i>item</i> , <i>format</i> and <i>key</i> . If <i>key</i> is the	

	last key for the <i>item/ format</i> pair, the advise loop for the pair is terminated.
	If <i>disconnectp</i> is t, and the last advise loop for the conversa- tion is terminated, the conversation is disconnected.
	Attempting to remove a link that does not exist raises an error, unless <i>no-advise-ok</i> is t.
	If this function succeeds, it returns t. If it fails, the behavior depends on the value of <i>errorp</i> . If errorp is t (the default value), LispWorks signals an error. If <i>errorp</i> is nil, the function returns nil to indicate failure.
See also	dde-advise-start
	dde-advise-start*
	dde-advise-stop*
	dde-client-advise-data

# dde-advise-stop\*

Summary	Removes a link from an automatically managed conversation specified by a given item and key.	
Package	win32	
Signature	dde-advise-sto disconnectp => r	op* service topic item &key key format errorp result
Arguments	service	A string or symbol.
	topic	A string or symbol.
	item	A string or symbol.
	key	An object.
	format	A clipboard format specifier.
	errorp	A boolean.

	disconnectp	A boolean.
Values	result	A boolean.
Description	The function dde-advise-stop* is similar to the function dde-advise-stop, and removes a particular link from a conversation specified by the <i>service/topic</i> pair indicated by <i>item</i> , <i>format</i> and <i>key</i> . If <i>key</i> is the last key for the <i>item/format</i> pair, the advise loop for the pair is terminated. If <i>disconnectp</i> is t (the default value), and the last advise loop for the conversation is terminated, the conversation is disconnected.	
	depends on the value), LispWor	succeeds, it returns t. If it fails, the behavior value of <i>errorp</i> . If errorp is t (the default rks signals an error. If <i>errorp</i> is nil, the func- to indicate failure.
See also	dde-advise-st	art
	dde-advise-st	art*
	dde-advise-st	op

## dde-client-advise-data

**Generic Function** 

Summary	Called when data changes in an advise loop.	
Package	win32	
Signature	dde-client-advise-data <i>key item data</i> &key &allow-other- keys =>	
Arguments	key	An object.
	item	A string or symbol.
	data	A string.

Values	None.
Description	The generic function dde-client-advise-data is the default function called when an advise loop informs a client that the data monitored by the loop has changed. By default it does nothing, but it may be specialized on the object used as the key in dde-advise-start Or dde-advise-start*, or on a cli- ent conversation class if the default <i>key</i> is used.
See also	dde-advise-start
	dde-advise-stop

## dde-connect

## Function

Summary	Attempts to create a conversation with a specified DDE server.		
Package	win32		
Signature	dde-connect service topic &key class errorp => object		
Arguments	service	A symbol or string.	
	topic	A symbol or string.	
	class	The class of the conversation object to create.	
	errorp	A boolean.	
Values	object	A conversation object.	
Description	The function dde-connect attempts to create a conversation with a DDE server. If <i>server</i> names a client service registered with define-dde-client, the registered service name is used as the DDE service name. If <i>server</i> is any other symbol, the print name of the symbol is used as the DDE service name. If		

print name of the symbol is used as the DDE service name. If *server* is a string, that string is used as the DDE service name.

The *topic* argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.

The *class* argument specifies the class of the conversation object to create. It must be a subclass of dde-clientconversation, or nil. If it is nil (the default value), then a conversation of class dde-client-conversation is created, unless *server* names a client service registered with definedde-client, in which case the registered class (if any) is used.

On executing successfully, this function returns a conversation object. If unsuccessful, the behavior depends on the value of *errorp*. If *errorp* is t (the default value), then an error is raised. If *errorp* is false, the function returns mil.

Note that conversation objects may only be used within the thread (lightweight process) in which they were created.

See also dde-disconnect

## dde-disconnect

Summary	Disconnects a conversation object.	
Package	win32	
Signature	dde-disconnect conversation => result	
Arguments	<i>conversation</i> A conversation object.	
Values	result A boolean.	
Description	The function dde-disconnect disconnects the conversation object. The conversation may no longer be used. If the conversation disconnects successfully, t is returned.	

See also dde-connect

## dde-execute

## Function

Summary	An alternative syntax for dde-execute-command.	
Package	win32	
Signature	dde-execute conversation command &rest {args}* => result	
Arguments	conversation	A conversation object.
	command	A string or symbol.
	args	An argument.
Values	result	A boolean.
Description	The function dde-execute provides an alternative syntax for dde-execute-command. Unlike dde-execute-command, dde-execute takes the arguments for <i>command</i> as a sequence of <i>args</i> following &rest, and does not have an argument for specifying how to handle an error.	
See also	dde-execute*	
	dde-execute-co	ommand*
	dde-execute-st	tring

## dde-execute\*

Summary	An alternative syntax for dde-execute-command*.		
Package	win32		
Signature	<pre>dde-execute* service topic command &amp;rest {args}* =&gt; result</pre>		

#### The WIN32 package (including DDE)

Arguments	service	A string or symbol.	
	topic	A string symbol.	
	command	A string or symbol.	
	args	An argument.	
Values	result	A boolean.	
Description	The function dde-execute* provides an alternative syntax for dde-execute-command*. Unlike dde-execute-command*, dde-execute* takes the arguments for <i>command</i> as a sequence of <i>args</i> following &rest, and does not have any arguments for specifying how to handle errors.		
See also	dde-execute		
	dde-execute-c	ommand	
	dde-execute-string		

## dde-execute-command

Summary	Sends a command string to a specified conversation.	
Package	win32	
Signature	dde-execute-co => <i>result</i>	mmand conversation command arg-list &key errorp
Arguments	conversation	A conversation object.
	command	A string or symbol.
	arg-list	A list of strings, integers, and floats.
	errorp	A boolean.
Values	result	A boolean.

Description The function dde-execute-command sends a command string to the conversation specified by *conversation*. The command string consists of *command* and *arg-list*, which are combined using the appropriate argument-marshalling conventions. By default, the syntax is

[command(arg1,arg2,...)]

On success, this function returns a result of t. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is t (the default value), LispWorks signals an error. If it is nil, the function returns nil to indicate failure.

See also dde-execute dde-execute-string

## dde-execute-command\*

#### Function

Summary	Sends a command string to a specified service on a given topic.		
Package	win32		
Signature	dde-execute-command* service topic command arg-list &key errorp conect-error-p new-conversation-p => result		
Arguments	service	A string or symbol.	
	topic	A string or symbol.	
	command	A string or symbol.	
	arg-list	A list of strings, integers, and floats.	
	errorp	A boolean.	
	connect-error-p	A boolean.	
	new-conversation-p		

A boolean.

Values	result	A boolean.
Description	cute-command, a specified by <i>ser</i> string consists o	e-execute-command* is similar to dde-exe- and sends a command string to the server vice on a topic given by <i>topic</i> . The command of <i>command</i> and <i>arg-list</i> , which are combined priate argument-marshalling conventions. By tax is
	[command(arg1,	arg2,)]
	client, the regivice name. If <i>ser</i> symbol is used	a client service registered with define-dde- stered service name is used as the DDE ser- ver is any other symbol, the print name of the as the DDE service name. If <i>server</i> is a string, ed as the DDE service name.
	in the conversat	ent specifies the DDE topic name to be used tion. If it is a symbol, the symbol's print name string, the string is used.
	conversation fo able conversation over that conver made with the s with-dde-conversation	e function dde-execute-command* creates a r the duration of the transaction, but if a suit- on already exists, the transaction is executed rsation. Hence, if several transactions will be same <i>service</i> and <i>topic</i> , placing them inside a ersation prevents a new conversation being each transaction.
	established for	<i>ion-p</i> is set to t a new conversation is always the transaction. This new conversation is tically disconnected when the transaction is
	cannot be established of the cannot be established of the cannot be established between the case	b is t (the default value) and a conversation lished, then LispWorks signals an error. If it is te-command* returns nil if a conversation lished. This allows the caller to distinguish ses when the server is not running, and when nning but the transaction fails.

Upon success, this function returns a result of t. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is t (the default value), LispWorks signals an error. If it is nil, the function returns nil to indicate failure.

See also dde-execute dde-execute-string dde-execute-command

## dde-execute-string

Summary	Issues an execute transaction consisting of a specified string.	
Package	win32	
Signature	dde-execute-string conversation command &key errorp => result	
Arguments	conversation	A conversation object.
	command	A string or symbol.
	errorp	A boolean.
Values	result	A boolean.
Description		le-execute-string issues an execute transac- of the string <i>command</i> . No processing of the ned.
Description	tion consisting string is perform On success, this depends on the default value), 1	of the string <i>command</i> . No processing of the
Description See also	tion consisting string is perform On success, this depends on the default value), 1	of the string <i>command</i> . No processing of the ned. If function returns t. On failure, the behavior value of the <i>errorp</i> argument. If <i>errorp</i> is t (the LispWorks signals an error. If it is nil, the

#### dde-execute-string\*

# dde-execute-string\*

Summary	Issues an execute transaction consisting of a specified string on an automatically managed conversation.		
Package	win32		
Signature	<pre>dde-execute-string* service topic command &amp;key errorp conect- error-p new-conversation-p =&gt; result</pre>		
Arguments	service	A symbol or string.	
	topic	A symbol or string.	
	command	A string or symbol.	
	errorp	A boolean.	
	connect-error-p	A boolean.	
	new-conversation	а-р	
		A boolean.	
Values	result	A boolean.	
Description	The function dde-execute-string* is similar to dde- execute-string, in that it issues an execute transaction sisting of the string <i>command</i> . However, the conversation across which <i>command</i> is issued is managed automatica No processing of the string is performed.		
	If server names a client service registered with define-dde- client, the registered service name is used as the DDE ser- vice name. If server is any other symbol, the print name of the symbol is used as the DDE service name. If server is a string, that string is used as the DDE service name.		

The *topic* argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.

If necessary, the function dde-execute-string\* will create a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction will be executed over that conversation. Hence, if several transactions will be made with the same *service* and *topic*, placing them inside a with-dde-conversation prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to t a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If connect-error-p is t (the default value), then LispWorks signals an error if a conversation cannot be established. If it is nil, dde-execute-string\* returns nil if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

Upon success, the function returns t. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is t (the default value), LispWorks signals an error. If it is nil, the function returns nil to indicate failure.

See also	dde-execute
	dde-execute-command
	dde-execute-string

## dde-item

#### Accessor

- Summary An accessor which can perform a request transaction or a poke transaction.
- Package win32
- Signature dde-item conversation item &key format type errorp => result

Arguments	conversation	A conversation object.
	item	A string or symbol.
	format	A clipboard format specifier.
	type	A keyword.
	errorp	A boolean.
Values	result	A boolean.

Description The accessor dde-item performs a request transaction when read. It performs a poke transaction when set.

To illustrate, the following dde-request command

(dde-request conversation item :format format :type type
:errorp errorp)

can also be issued using dde-item as follows:

(dde-item conversation item :FORMAT format :TYPE type :ERRORP errorp)

Similarly, the following dde-poke command

(dde-poke conversation item data :format format :type type<br/>:errorp errorp)can be issued using dde-item as follows:<br/>(setf (dde-item conversation item :format format :type type<br/>:errorp errorp) data)<br/>except that the format always returns data.<br/>Upon success, this function returns a result of t. On failure,<br/>the behavior depends on the value of the errorp argument. If<br/>errorp is t (the default value), LispWorks signals an error. If it<br/>is nil, the function returns nil to indicate failure.See alsodde-item\*<br/>dde-poke<br/>dde-request

dde-item*		Accessor
Summary	An accessor which can perform a request transaction or a poke transaction on an automatically managed conversation.	
Package	win32	
Signature	<pre>dde-item* service topic item &amp;key format type errorp connect- error-p new-conversation-p =&gt; result</pre>	
Arguments	service	A string or symbol.
	topic	A string or symbol.
	item	A string or symbol.
	format	A clipboard format specifier.
	type	A keyword.
	errorp	A boolean.
	connect-error-p	A boolean.

#### new-conversation-p

completed.

A boolean.

Values	result	A boolean.
Description	The accessor dde-item* is similar to dde-item, and perform a request transaction when read. It performs a poke transac tion when set.	
	To illustrate, th	e following dde-request* command
		service topic item :format format :type type connect-error-p new-conversation-p)
	can also be issu	ned using dde-item* as follows:
		vice topic item :FORMAT format :TYPE type connect-error-p new-conversation-p)
	Similarly, the fo	ollowing dde-poke* command
		versation item data :format format :type type connect-error-p new-conversation-p)
	can be issued using dde-item* as follows:	
		em* conversation item :format format :type type connect-error-p new-conversation-p) data)
	except that the	<i>format</i> always returns <i>data</i> .
	for the duration tion already ex versation. If yo same <i>service</i> an	e accessor dde-item* creates a conversation n of the transaction, but if a suitable conversa- ists, the transaction is executed over that con- u need to make several transactions with the d <i>topic</i> , placing them inside a with-dde- prevents a new conversation being established ction.
	established for	<i>tion-p</i> is set to t a new conversation is always the transaction. This new conversation is tically disconnected when the transaction is

	If connect-error-p is t (the default value), then LispWorks sig- nals an error if a conversation cannot be established. If it is nil, dde-item* returns nil if a conversation cannot be estab- lished. This allows the caller to distinguish between the cases when the server is not running, and when the server is run- ning but the transaction fails.			
	On success, the function returns t. On failure, the behavior depends on the value of the <i>errorp</i> argument. If <i>errorp</i> is t (the default value), LispWorks signals an error. If it is nil, the function returns nil to indicate failure.			
See also	dde-item			
	dde-poke			

dde-request

dde-poke		Function
Summary	Issues a poke transaction on a conversation, to set the value of a specified item.	
Package	win32	
Signature	dde-poke conve result	ersation item data &key format type errorp =>
Arguments	conversation	A conversation object.
	item	A string or symbol.
	data	A string.
	format	A clipboard format specifier.
	type	A keyword.
	errorp	A boolean.
Values	result	A boolean.

Description	The function dde-poke issues a poke transaction on conversa-
	tion to set the value of the item specified by item to the value
	specified by <i>data</i> . The argument <i>item</i> should be a string, or a
	symbol. If it is a symbol its print name is used.

The argument *format* should be one of the following:

- A DDE format specifier, consisting of either a standard clipboard format or a registered clipboard format.
- A string containing either the name of a standard clipboard format (without the CF\_ prefix), or the name of a registered clipboard format.
- A symbol, in which case its print name is taken to specify the clipboard format.
- The keyword :text. This is the default value.

The keyword :text is treated specially. If supported by the server it uses the CF\_UNICODETEXT clipboard format, otherwise it used the CF\_TEXT format.

For text transactions, the default value of *type* indicates that *data* is a Lisp string to be used. If *type* is :string-list, then *data* is taken to be a list of strings, and is sent as a tab-separated string.

Alternatively, *data* can be a clipboard-item structure, containing a foreign pointer to the data to send and the length of the data. In this case the *type* argument is ignored.

On success, this function returns t. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is t (the default value), LispWorks signals an error. If it is nil, the function returns nil to indicate failure.

See also dde-item

dde-request

## dde-poke\*

#### Function

Summary	Issues a poke transaction on an automatically managed conversation, to set the value of a specified item.		
Package	win32		
Signature	dde-poke* service topic item data &key format type errorp connect-error-p new-conversation-p => result		
Arguments	service	A symbol or string.	
	topic	A symbol or string.	
	item	A string or symbol.	
	data	A string.	
	format	A clipboard format specifier.	
	type	A keyword.	
	errorp	A boolean.	
	connect-error-p	A boolean.	
	new-conversation-p		
		A boolean.	
Values	result	A boolean.	
Description	The function dde-poke* is the same as dde-poke, except that conversations are managed automatically. The function issues a poke transaction to set the value of the item specified by <i>item</i> to the value specified by <i>data</i> . The argument <i>item</i> should be a string, or a symbol. If it is a symbol its print name		

is used.

If server names a client service registered with define-ddeclient, the registered service name is used as the DDE service name. If server is any other symbol, the print name of the symbol is used as the DDE service name. If *server* is a string, that string is used as the DDE service name.

The *topic* argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.

For information on the *format, type*, and *errorp* arguments, see dde-poke.

If necessary, the function dde-poke\* creates a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction is executed over that conversation. Hence, if several transactions are made with the same *service* and *topic*, placing them inside a with-ddeconversation prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to t a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If connect-error-p is t (the default value), LispWorks signals an error if a conversation cannot be established. If it is nil, dde-poke\* returns nil if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

See also dde-item

dde-request

#### dde-request

#### Function

Summary	Issues a request transaction on a conversation for a specified item.	
Package	win32	
Signature	dde-request conversation item &key format type errorp => result successp	
Arguments	conversation	A conversation object.
	item	A string or symbol.
	format	A clipboard format specifier.
	type	A keyword.
	errorp	A boolean.
Values	result	The return value of the transaction.
	successp	A boolean.
Description	The function dde-request issues a request transaction on	

escription The function dde-request issues a request transaction on conversation for the specified *item*. The argument *item* should be a string, or a symbol. If it is a symbol its print name is used.

The argument *format* should be one of the following:

- A DDE format specifier, consisting of either a standard clipboard format or a registered clipboard format.
- A string containing either the name of a standard clipboard format (without the CF\_ prefix), or the name of a registered clipboard format.

- A symbol, in which case its print name is taken to specify the clipboard format.
- The keyword :text. This is the default value.

The keyword :text is treated specially. If supported by the server it uses the CF\_UNICODETEXT clipboard format, otherwise it used the CF\_TEXT format.

The default conversation class only supports text formats, unless *type* is specified as :foreign. The argument *type* specifies how the response data should be converted to a Lisp object. For text formats, the default value indicates that a Lisp string should be created. The value :string-list may be specified for *type* to indicate that the return value should be taken as a tab-separated list of strings; in this case the Lisp return value is a list of strings. The value :foreign can be used with any clipboard format. It returns a clipboard-item structure, containing a foreign pointer to the data, the data length, and the format identifier.

This function returns two values, *result* and *successp*. If successful, *result* is the return value of the transaction (which may be nil in the case of :string-list), and *successp* is true to indicate success.

On failure, the result of the function depends on the *errorp* argument. If *errorp* is t (the default), the function signals an error. If *errorp* is nil, the function returns (values nil nil).

See also dde-item

dde-poke

dde-request\*

## dde-request\*

## Function

Summary Issues a request transaction on an automatically managed conversation for a specified item.

Package	win32	
Signature	dde-request* service topic item &key format type errorp connect- error-p new-conversation-p => result successp	
Arguments	service	A symbol or string.
	topic	A symbol or string.
	item	A string or symbol.
	format	A clipboard format specifier.
	type	A keyword.
	errorp	A boolean.
	connect-error-p	A boolean.
	new-conversation-p	
		A boolean.
Values	result	The return value of the transaction.
Description	The function dde-request* is similar to dde-request, except that conversations are managed automatically. The function issues a request transaction for the specified <i>item</i> . The argument <i>item</i> should be a string, or a symbol. If it is a symbol its print name is used.	
	If server names a client service registered with define-dde- client, the registered service name is used as the DDE ser- vice name. If server is any other symbol, the print name of the symbol is used as the DDE service name. If server is a string, that string is used as the DDE service name.	
	The <i>topic</i> argument specifies the DDE topic name to be use in the conversation. If it is a symbol, the symbol's print nar is used. If it is a string, the string is used.	

For information on the *format, type*, and *errorp* arguments see dde-request.

If necessary, the function dde-request\* will create a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction will be executed over that conversation. Hence, if several transactions will be made with the same *service* and *topic*, placing them inside a with-dde-conversation prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to t a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If connect-error-p is t (the default value), then LispWorks signals an error if a conversation cannot be established. If it is nil,dde-request\* returns nil if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

See also	dde-item
----------	----------

dde-poke

dde-request

## define-dde-client

Summary	Registers a clie	ent service.
Package	win32	
Signature	define-dde-cl	ient name &key service class => name
Arguments	name	A symbol.
	service	A string.
	class	A subclass of dde-client-conversation

Values *name* A symbol.

Description The macro define-dde-client defines a mapping from the symbol *name* to the DDE service name with which to establish a conversation, and the conversation class to use for this conversation. The argument *service* is a string which names the DDE service. It defaults to the print-name of *name*. The argument *class* is a subclass of dde-client-conversation which is used for all conversations with this service. It defaults to dde-client-conversation. Specifying a subclass allows various aspects of the behavior of the conversation to be specialized.

Note that it is generally not necessary to register client services unless a specialized conversation type is required. However, it is sometimes convenient to register a client service in order to allow the service name to be changed in the future.

If the macro executes successfully, the *name* of the DDE service is returned.

See also dde-connect dde-disconnect

with-dde-conversation

## with-dde-conversation

Summary	Dynamically binds a conversation to a server across a given body of code.
Package	win32
Signature	with-dde-conversation ( <i>conv service topic</i> &key errorp new-conversation-p) &body body => result
Arguments	<i>conv</i> A conversation object.

	service	A symbol or string.	
	topic	A symbol or string.	
	errorp	A boolean.	
	new-conversation	n-p	
		A boolean.	
	body	A list of Lisp forms.	
Values	result	A boolean.	
Description	The macro with-dde-conversation dynamically binds a conversation with a server across the scope of a body of code specified by <i>body</i> . The argument <i>conv</i> is bound to a conversation with the server specified by <i>service</i> , and the topic specified by <i>topic</i> .		
	If server names a client service registered with define-dde- client, the registered service name is used as the DDE ser- vice name. If server is any other symbol, the print name of the symbol is used as the DDE service name. If server is a string, that string is used as the DDE service name.		
	The <i>topic</i> argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.		
	An existing conversation may be used, if available, unless <i>new-conversation-p</i> is true, in which case a new conversation is always created.		
	If a new conversation is created, it is disconnected after <i>body</i> has executed as an implicit program.		
	the function de default value),	n cannot be established, the result returned by pends on the value of <i>errorp</i> . If <i>errorp</i> is t (the then LispWorks signals an error. If <i>errorp</i> is a not executed, and nil is returned.	

See also define-dde-client

## **18.3 DDE server interface reference entries**

## dde-server-poke

## **Generic Function**

Summary	Called when a poke transaction is received.		
Package	win32		
Signature	dde-server-poke <i>server topic item data</i> &key <i>format</i> &allow- other-keys => <i>successp</i>		
Arguments	server	A server object.	
	topic	A topic object.	
	item	A string.	
	data	A string.	
	format	A keyword.	
Values	successp	A boolean.	
Description	The generic function dde-server-poke is called in response to a poke transaction. A method specializing on the classes of <i>server</i> and <i>topic</i> should poke the data given by <i>data</i> into the item specified by <i>item</i> .		
	The keyword <i>format</i> indicates the format in which the item is being requested. By default, only text transfers are supported (and the <i>format</i> argument will have the value :text). The set of supported formats may be extended in future releases, so applications should always check the value of the format parameter and reject transactions which use formats not supported by the application.		

If the poke transaction is successful, non-nil should be returned, and nil should be returned for failure.

See also dde-poke dde-request dde-server-request

## dde-server-request

Generic Function

Summary	Called when a request transaction is received.			
Package	win32	win32		
Signature		dde-server-request <i>server topic item</i> &key <i>format</i> &allow- other-keys => <i>data</i>		
Arguments	server	A server object.		
	topic	A topic object.		
	item	A string.		
	format	A keyword.		
Values	data	The returned data.		
response to a request transaction. A m		function dde-server-request is called in request transaction. A method specializing on f <i>server</i> and <i>topic</i> should return the data in <i>item</i> .		
	The expected format of the data is given by <i>format</i> , which defaults to :text. The set of supported formats may be extended in future releases, so applications should always check the value of the format parameter and reject transactions which use formats not supported by the application.			
	If the request fails, <code>lii</code> should be returned.			

See also dde-poke dde-request dde-server-poke

## dde-server-topic

**Generic Function** 

Summary Called whenever a client attempts to connect to a server with a given topic. Package win32 Signature dde-server-topic server topic-name => topic Arguments server A server. topic-name A string. Values topic A topic. Description The generic function dde-server-topic is called whenever a client attempts to make a connection to the server. The argument *topic-name* is a string identifying a topic. If the server recognizes the topic, a method specializing on the server should return an instance of one of the server's topic classes. If the server does not recognize the topic, the method should return nil. See also dde-server-topics dde-topic-items

## dde-server-topics

**Generic Function** 

Summary Returns a list of the available general topics on a given server.

#### The WIN32 package (including DDE)

	Package	win32		
	Signature	dde-server-top:	ics server => topic-list	
	Arguments	server	A server object.	
	Values	topic-list	A list of strings.	
	Description	available general specializing on t ing topics (see a returned, as the you do not prov default method	ction dde-server-topics al topics on a given server, the server class should be efine-dde-dispatch-top y are handled automatical ride a dde-server-topics returns :unknown, which p oonding to the topics requ	A suitable method defined. Dispatch- bic) should not be lly by LispWorks. If method, the prevents the DDE
		each topic, even alternative form application imp foo, foo.doc an for referring to t	one canonical name shoul though the server may re is of name for a topic. For lements a topic for each o d c:\foo.doc may all be a he same topic; however a ach topic once only.	ecognize several example, if an pen file, the topics acceptable strings
		The application server-topic g	must also provide a meth eneric function.	od on the dde-
	See also	dde-server-top dde-topic-item		
C	Ide-topic-item	IS		Generic Function

Summary	Returns the valid items in a topic.
---------	-------------------------------------

Package win32

Signature	dde-topic-items server topic => item-strings	
Arguments	server	A server object.
	topic	A topic object.
Values	item-strings	A list of strings.
Description	The generic function dde-topic-items returns a list of strings corresponding to the valid items in the topic. A method specializing on a server and topic should be defined.	
	If it is not practical to return a list of the items (for example, if the list is potentially infinite), the generic function returns :unknown.	
See also	dde-server-topic	
	dde-server-topics	

## define-dde-dispatch-topic

Summary	Defines a dispatch topic.	
Package	win32	
Signature	define-dde-dis name	patch-topic name &key server topic-name =>
Arguments	name	A symbol.
	server	A server class.
	topic-name	A string.
Values	name	A symbol.
Description	The macro define-dde-dispatch-topic defines a dispatch- ing topic. A dispatching topic is a topic which has a fixed	

	name and always exists. Dispatching topics provide dis- patching capabilities, whereby appropriate application-sup- plied code is executed for each supported transaction. Note that the server implementation also provides some dispatch- ing capabilities.
	The name of the dispatching topic object is specified by <i>name</i> .
	The topic is identified by the string <i>topic-name</i> .
	The class of the server to attach the topic to is given by server.
	The macro define-dde-dispatch-topic returns the name of the dispatching topic, <i>name</i> .
See also	dde-server-topic
	dde-server-topics

## define-dde-server

Summary	Defines a class for a Lisp DDE server.	
Package	win32	
Signature	<pre>define-dde-server class-name service-name =&gt; class-name</pre>	
	<pre>define-dde-server class-name superclasses slot-specs options =&gt; class-name</pre>	
Arguments	class-name	A class name.
	service-name	A string.
	superclasses	A list of superclasses.
	slot-specs	The specifications for the class' slots.
	options	A keyword option.
Values	class-name	A class name.

Description	The macro define-dde-server defines a class for a Lisp DDE server. The class inherits from dde-server.		
	The long form of the macro is similar to defclass, but with one extra option, :service, which is used to specify the ser- vice name string to which this server will respond.		
	The short form is provided to handle the common simple case; <i>class-name</i> is the name of the Lisp class to be defined, and <i>service-name</i> is the service name string to which this server will respond.		
Example	The first example uses the short version of define-dde- server to define a class, called lisp-server, which has the service name "LISP".		
	(define-dde-server lisp-server "LISP")		
	The second example shows how to use the long for of the macro to define the same class, and illustrates the use of the <i>superclasses</i> and <i>options</i> arguments.		
	<pre>(define-dde-server lisp-server (dde-server)   ()   (:service "LISP"))</pre>		
See also	dde-server-topic		
	dde-server-topics		

## define-dde-server-function

dde-topic-items

Summary	Defines a server function that is called when a specific trans- action occurs.
Package	win32
Signature	<pre>define-dde-server-function name-and-options transaction (binding*) form* =&gt; name</pre>

```
name-and-options ::= name | (name [[option]])
transaction ::= :request | :poke | :execute
option ::= :server server |
            :topic-class topic-class |
            :topic topic |
            :item item |
            :format format |
            :command command |
            :result-type result-type |
            :advisep advisep
binding ::= var-binding | execute-arg-binding
var-binding ::= (var :server) |
                  (var :topic)
                  (var :data [data-type])
                  (var :format)
execute-arg-binding ::= var | (var type-spec)
```

Arguments	name	A symbol.
	transaction	A keyword.
	server	A server object.
	topic-class	A topic class.
	topic	A topic object.
	item	A string.
	format	A keyword.
	command	A string.
	result-type	A data type.
	advisep	A boolean.
	var	A variable.
	data-type	A data type.
	type-spec	A data type.
	form	A Lisp form.

Values name A symbol.

Description The macro define-dde-server-function is used to define a server function, called *name*, which is called when a specific transaction occurs.

The defined function may either be attached to a server object (possibly only for a particular topic class) or to a dispatching topic object.

To attach the definition to a server, :server should be used to specify the server class. :topic-class may be used to specify the topic-class for which this definition should be used. It can be a symbol which names a topic-class, or one of the following special symbols:

t	All topics (default for execute transactions).
:system	The System topic.
:non-system	Any topic except the system topic.

In the case of execute transactions only, :topic-class defaults to t; in all other cases, it must be specified. Typically, execute transactions ignore the topic of the conversation. Alternatively, you may choose to only support execute transactions in the system topic.

A server function may instead be attached to a particular instance of dde-dispatch-topic, previously defined by define-dde-dispatch-topic. This is the main use of dispatching topics.

In this case :topic should be provided with a symbol that names a dispatching topic. The function is installed on that topic, and only applies to that topic.

In the case of a request or poke transaction, *item* is a string defining the item name for which this definition should be invoked. It defaults to the capitalized print-name of *name*, with hyphens removed.

For request transactions, the :format option is used to specify the format understood. It defaults to :text. It can be specified as :all, in which case the :format binding may be used to determine the actual format requested (see below).

In the case of an execute transaction, *command* is a string specifying the name of the command for which this definition should be invoked. It defaults to the capitalized printname of *name*, with hyphens removed.

The *execute-arg-bindings* are only used with execute transactions. They specify the arguments expected. *type-spec* should be one of t, string, number, integer or float. If not specified, t is assumed.

The var-bindings may appear anywhere in the binding list, and in any order. Binding variables to :server and :topic is useful with all transaction types. A :server binding causes the variable to be bound to the server object, whereas a :topic binding causes the variable to be bound to the topic object. This allows the server and/or the topic to be referred to in the body of the function.

A :format binding can only be used with request and poke transactions, where an *option* of :format :all has been specified. It causes the variable specified by *var* to be bound to the format of data requested or supplied. The body of the defined function should fail the transaction if it does not support the requested format.

A :data binding can only be used with poke transactions. It binds a variable to the data to be poked. For text transfers, the data variable is normally bound to a string. However, if *data-type* is specified as :string-list, the data in the transaction is interpreted as a tab-separated list of strings, and the data variable is bound to a list of strings.

For execute and poke transactions, the body of the defined function is expected to return t for success and mil for failure.

For request transactions, the body of the defined function is normally expected to return a result value, or nil for failure.

The *result-type* option may only be specified for request transactions. If it is specified as <code>:string-list</code>, then for text requests the body is expected to return a list of strings, which are used to create a tab-separated list to be returned to the client.

Sometimes, it may be necessary to support returning nil to mean the empty list, rather than failure. In this case, the result-type can be specified as (:string-list t). The body is then expected to return two values: a list of strings, and a flag indicating success.

In the case of execute transactions, the command name and arguments are unmarshalled by the default argument unmarshalling. This is compatible with the default argument unmarshalling described under dde-execute-command. The execute string is expected to be of the following syntax:

```
[command1(arg1,arg2,...)][command2(arg1,arg2,...)]...]
```

Note that multiple commands may be packed into a single execute transaction. However, dde-execute-command does not currently generate such strings.

See also dde-execute-command define-dde-client define-dde-dispatch-topic define-dde-server

## start-dde-server

#### Function

Summary Creates and starts an instance of a DDE server.

Package win32

#### The WIN32 package (including DDE)

Signature	<pre>start-dde-server name =&gt; server</pre>	
Arguments	name	A DDE server class
Values	server	A server object
Description	The function start-dde-server creates an instance of a server of the class specified by <i>name</i> which then starts accepting transactions. If successful the function returns the server, otherwise nil is returned.	
See also	define-dde-se	rver

# Index

#### Symbols

\*coff-loading-verbose\* 158 \*load-source-if-newer\* 197 \*load-when-compile\* 166

#### **Numerics**

16-bit-string type 160 8-bit-string type 159

#### A

accessor dde-item 380 dde-item\* 381 \*active-finders\* variable 87 add-special-free-action function 95 add-sql-stream function 269 add-symbol-profiler function 96 allocation-in-gen-num macro 96 appendf macro 160 apropos-color-alias-names function 9 apropos-color-names function 10 apropos-color-spec-names function 11 attribute-type function 270 augmented-string type 343 augmented-string-p function 343 avoid-gc function 98

#### B

base-char type 160 base-character type 161 base-character-p function 161 base-char-code-limit constant 162 base-char-p function 162 base-string type 35 

#### С

call-next-advice function 163 calls-who function 98 change-directory function 99 character type 36 characterp function 36 char-external-code function 89 claim-lock function 217 classes fundamental-binary-inputstream 321 fundamental-binary-outputstream 322 fundamental-binary-stream 322 fundamental-character-inputstream 323 fundamental-character-outputstream 324 fundamental-character-stream 324 fundamental-input-stream 325 fundamental-output-stream 326 fundamental-stream 326 socket-stream 33 clean-down function 99 clean-generation-0 function 100

close generic function 37 coerce function 37 collect-generation-2 function 101 collect-highest-generation function 101 color-<component> function 12 \*color-database\* variable 13 color-level function 13 color-model function 14 colors= function 15 commit function 271 compile function 38 compile-file function 39 compile-system function 164 concatenate function 42 concatenate-system function 166 conditions external-format-error 90 file-encoding-resolutionerror 350 connect function 272 connected-databases function 274 \*connect-if-exists\* variable 273 constants base-char-code-limit 162 convert-color function 15 create-index function 275 create-process function 218 create-table function 275 create-view function 276 create-view-from-class function 277 \*current-process\* special variable 222 current-stack-length function 102

#### D

database-name function 278 dde-advise-start function 364 dde-advise-start\* function 366 dde-advise-stop function 368 dde-advise-stop\* function 369 dde-client-advise-data generic function 370 dde-connect macro 371 dde-disconnect macro 372 dde-execute function 373 dde-execute\* function 373 dde-execute-command function 374 dde-execute-command\* function 375 dde-execute-string function 377 dde-execute-string\* function 378 dde-item accessor 380

dde-item\* accessor 381 dde-poke function 383 dde-poke\* function 385 dde-request function 387 dde-request\* function 388 dde-server-poke generic function 393 dde-server-request generic function 394 dde-server-topic generic function 395 dde-server-topics generic function 395 dde-topic-items generic function 396 \*debug-print-length\* variable 77 \*debug-print-level\* variable 78 declare special form 42 defadvice macro 168 \*default-action-list-sort-time\* variable 171 \*default-character-element-type\* parameter 171 \*default-database\* variable 279 \*default-database-type\* variable 279 default-eol-style function 344 default-file-encoding function 345 \*default-libraries\* variable 157 \*default-package-use-list\* variable 102 \*default-process-priority\* variable 222 \*default-profiler-limit\* variable 103 \*default-simple-process-priority\* variable 222 define-action macro 172 define-action-list macro 174 define-color-alias function 16 define-color-models macro 17 define-dde-client macro 390 define-dde-dispatch-topic macro 397 define-dde-server macro 398 define-dde-server-function macro 399 defpackage macro 44 defparser macro 263 defsystem macro 175 deftransform macro 73 def-view-class macro 279 delete-advice function 103 delete-color-translation function 18 delete-instance-records generic function 285 delete-records function 286 delete-sql-stream function 287 detect-encoding-in-file function 345 detect-eol-style function 346 detect-japanese-encoding-in-file

function 347 directory function 45 disable-sql-reader-syntax function 287 disassemble function 46 disconnect function 288 do-nothing function 180 do-query macro 289 dotted-list function 179 dotted-list-p function 180 drop-index function 290 drop-table function 290 drop-view function 291 drop-view-from-class function 291 dump-form function 105 dump-forms-to-file function 105

#### Е

enable-sql-reader-syntax function 292 enlarge-static function 107 ensure-<command> function 19 ensure-process-cleanup function 223 environment-variable function 181 execute-actions macro 182 execute-command function 293 expand-generation-1 function 108 extend-current-stack function 108 extended-char type 184 extended-character type 184 extended-character-p function 184 extended-char-p function 185 \*extended-spaces\* variable 213, 348 extended-time function 109 external-format-error condition 90 external-format-foreign-type function 91 \*external-formats\* variable 185 external-format-type function 91

#### F

find-encoding-option function 351 find-external-char function 92 find-filename-pattern-encodingmatch function 352 find-object-size function 110 find-process-from-name function 224 flag-not-special-free-action function 111 flag-special-free-action function 112 foreign-symbol-address function 150 function-lambda-list function 187 functions add-special-free-action 95 add-sql-stream 269 add-symbol-profiler 96 apropos-color-alias-names 9 apropos-color-names 10 apropos-color-spec-names 11 attribute-type 270 augmented-string-p 343 avoid-gc 98 base-character-p 161 base-char-p 162 base-string-p 162 break-new-instances-on-access 1 break-on-access 2 break-on-unresolved 149 call-next-advice 163 calls-who 98 change-directory 99 characterp 36 char-external-code 89 claim-lock 217 clean-down 99 clean-generation-0 100 coerce 37 collect-generation-2 101 collect-highest-generation 101 color-<component> 12 color-level 13 color-model 14 colors= 15 commit 271 compile 38 compile-file 39 compile-system 164 concatenate 42 concatenate-system 166 connect 272 connected-databases 274 convert-color 15 create-index 275

```
create-process 218
create-table 275
create-view 276
create-view-from-class 277
current-stack-length 102
database-name 278
dde-advise-start 364
dde-advise-start* 366
dde-advise-stop 368
dde-advise-stop* 369
dde-execute 373
dde-execute* 373
dde-execute-command 374
dde-execute-command* 375
dde-execute-string 377
dde-execute-string* 378
dde-poke 383
dde-poke* 385
dde-request 387
dde-request* 388
default-eol-style 344
default-file-encoding 345
define-color-alias 16
delete-advice 103
delete-color-translation 18
delete-records 286
delete-sql-stream 287
detect-encoding-in-file 345
detect-eol-style 346
detect-japanese-encoding-in-
    file 347
directory 45
disable-sql-reader-syntax 287
disassemble 46
disconnect 288
do-nothing 180
dotted-list 179
dotted-list-p 180
drop-index 290
drop-table 290
drop-view 291
drop-view-from-class 291
dump-form 105
dump-forms-to-file 105
enable-sql-reader-syntax 292
enlarge-static 107
ensure-<command> 19
ensure-process-cleanup 223
environment-variable 181
execute-command 293
expand-generation-1 108
extend-current-stack 108
```

extended-character-p 184 extended-char-p 185 extended-time 109 external-format-foreign-type 91 external-format-type 91 false 186 file-directory-p 186 find-database 293 find-encoding-option 351 find-external-char 92 find-filename-pattern-encodingmatch 352 find-object-size 110 find-process-from-name 224 flag-not-special-free-action 111 flag-special-free-action 112 foreign-symbol-address 150 function-lambda-list 187 gc-if-needed 112 get-all-color-names 20 get-color-alias-translation 21 get-color-spec 22 get-default-generation 113 get-foreign-symbol 151 get-gc-parameters 187 get-working-directory 114 guess-external-format 353 hardcopy-system 193 initialize-database-type 294 initialize-multiprocessing 224 insert-records 295 interactive-stream-p 48 lisp-name-to-foreign-name 152 list-all-processes 225 list-attributes 297 list-classes 297 list-sql-streams 298 list-tables 299 load-all-patches 195 load-color-database 23 load-logical-pathnametranslations 48 load-system 196 locally-disable-sql-readersyntax 299 locally-enable-sql-readersyntax 300 lock-name 226 lock-owner 226 mailbox-empty-p 227 mailbox-peek 228 mailbox-read 228

mailbox-reader-process 229 mailbox-send 229 make-gray 23 make-hsv 24 make-lock 230 make-mailbox 231 make-named-timer 232 make-rgb 25 make-sequence 50 make-timer 233 make-unregistered-action-list 197 map 51 map-all-processes 234 map-all-processes-backtrace 235 map-process 236 map-process-backtrace 235 map-query 302 mark-and-sweep 355 merge 51 merge-ef-specs 356 normal-gc 117 notice-fd 236 object-address 357 open 52 open-tcp-stream 29 pointer-from-address 358 print-action-lists 199 print-actions 199 print-profile-list 118 print-query 303 process-alive-p 237 process-allow-scheduling 238 process-arrest-reasons 238 process-break 239 process-foreign-file 153 process-interrupt 239 process-kill 240 process-lock 240 process-name 241 process-p 242 process-plist 242 process-priority 243 process-reset 243 process-run-function 244 process-run-reasons 244 process-send 245 process-unlock 246 process-wait 246 process-wait-function 247 process-wait-with-timeout 248 proclaim 55 ps 248

query 305 read-color-db 26 read-foreign-modules 157 release-lock 249 remove-advice 201 remove-special-free-action 123 remove-symbol-profiler 123 reset-profiler 124 restore-sql-reader-syntaxstate 305 rollback 306 room 56 save-image 125 save-tags-database 88 sbchar 203 schedule-timer 249 schedule-timer-milliseconds 251 schedule-timer-relative 252 schedule-timer-relativemilliseconds 254 select 307 set-default-character-elementtype 204 set-default-generation 128 setf timer-name 255 set-gc-parameters 129 set-minimum-free-space 131 set-profiler-threshold 132 set-promotion-count 132 set-up-profiler 133 simple-augmented-string-p 360 simple-base-string-p 204 simple-char-p 205 simple-process-p 256 simple-text-string-p 206 source-debugging-on-p 134 sal 309 sql-expression 309,311 sql-operator 312 sql-recording-p 312 sgl-stream 313 start-dde-server 403 start-sql-recording 314 start-up-server 31 staticp 361 status 315 stchar 207 stop-sql-recording 316 string-append 207 switch-static-allocation 135 text-string-p 208 timer-expired-p 256

timer-name 257 toggle-source-debugging 136 total-allocation 137 trace-new-instances-on-access 3 trace-on-access 4 true 210 unbreak-new-instances-onaccess 5 unbreak-on-access 6 unnotice-fd 258 unschedule-timer 259 untrace-new-instances-onaccess 6 untrace-on-access 7 update-records 316 valid-external-format-p 93 whitespace-char-p 212 who-calls 145 with-output-to-fasl-file 146 yield 262 fundamental-binary-input-stream class 321 fundamental-binary-output-stream class 322 fundamental-binary-stream class 322 fundamental-character-inputstream class 323 fundamental-character-outputstream class 324 fundamental-input-stream class 324, 325 fundamental-output-stream class 326 fundamental-stream class 326

#### G

gc-if-needed function 112 generic functions close 37 dde-client-advise-data 370 dde-server-poke 393 dde-server-request 394 dde-server-topic 395 dde-server-topics 395 dde-topic-items 396 delete-instance-records 285 get-inspector-values 188 input-stream-p 47 open-stream-p 54 output-stream-p 54 stream-advance-to-column 327 stream-clear-input 328 stream-clear-output 328

stream-element-type 63 stream-finish-output 329 stream-force-output 329 stream-fresh-line 330 stream-line-column 330 stream-listen 331 stream-output-width 331 stream-peek-char 332 stream-read-byte 333 stream-read-char 333 stream-read-char-no-hang 334 stream-read-line 335 stream-read-sequence 335 stream-start-line-p 336 stream-terpri 337 stream-unread-char 338 stream-write-byte 338 stream-write-char 339 stream-write-sequence 340 stream-write-string 341 update-record-from-instance 317 update-record-from-slot 318 update-record-from-slots 319 get-all-color-names function 20 get-color-alias-translation function 21 get-color-spec function 22 get-default-generation function 113 get-foreign-symbol function 151 get-gc-parameters function 187 get-inspector-values generic function 188 get-working-directory function 114 guess-external-format function 353

#### Η

\*handle-existing-action-in-actionlist\* variable 191 \*handle-existing-action-list\* variable 191 \*handle-existing-defpackage\* variable 115 \*handle-mssing-action-in-actionlist\* variable 192 \*handle-missing-action-list\* variable 192 \*handle-warn-on-redefinition\* variable 193 hardcopy-system function 193 \*hidden-packages\* variable 79

#### I

#### L

library formats 157 \*line-arguments-list\* variable 355 link-load package symbols break-on-unresolved-functions 149 foreign-symbol-address 150 get-foreign-symbol 151 lisp-name-to-foreign-name 152 process-foreign-code 152 process-foreign-file 153 read-foreign-modules 157 lisp-name-to-foreign-name function 152 lispworks package symbols compile-system 164 defsystem 176 load-system 193, 196 \*lispworks-directory\* variable 194 list-all-processes function 225 list-attributes function 297 list-classes function 297 list-sgl-streams function 298 list-tables function 299 load-all-patches function 195 load-color-database function 23 load-logical-pathname-translations function 48 \*load-source-if-newer\* 197 load-system function 196 \*load-when-compile\* 166 locally-disable-sql-reader-syntax function 299 locally-enable-sql-reader-syntax function 300 lock-name function 226 lock-owner function 226 100p macro 49, 301

#### Μ

macros allocation-in-gen-num 96 appendf 160 dde-connect 371 dde-disconnect 372 defadvice 168 define-action 172 define-action-list 174 define-color-models 17 define-dde-client 390 define-dde-dispatch-topic 397 define-dde-server 398 define-dde-server-function 399 defpackage 44 defparser 263 defsystem 175 deftransform 73 def-view-class 279 do-query 289 execute-actions 182 in-static-area 354 100p 49.301 process-foreign-file 152 profile 120 rebinding 200 removef 202 step 59 time 64 top-level-form 209 trace 65 undefine-action 210 undefine-action-list 211 untrace 71 when-let 212 with-action-item-errorhandling 213 with-action-list-mapping 214 with-dde-conversation 391 with-heavy-allocation 146 with-lock 260 without-interrupts 261 without-preemption 261 with-transaction 319 with-unique-names 216 mailbox-empty-p function 227 mailbox-peek function 228 mailbox-read function 228 mailbox-reader-process function 229 mailbox-send function 229 make-gray function 23 make-hsv function 24

make-lock function 230 make-mailbox function 231 make-named-timer function 232 make-rgb function 25 make-sequence function 50 make-timer function 233 make-unregistered-action-list function 197 map function 51 map-all-processes function 234 map-all-processes-backtrace function 235 map-process function 236 map-process-backtrace function 235 map-query function 302 mark-and-sweep function 355 \*max-trace-indent\* variable 116 merge function 51 merge-ef-specs function 356 \*multibyte-code-page-ef\* variable 363

#### Ν

normal-gc function 117 notice-fd function 236

#### 0

object-address function 357 open function 52 open-stream-p generic function 54 open-tcp-stream function 29 output-stream-p generic function 54

#### P

\*packages-for-warn-on-redefinition\* variable 118 parameters \*default-character-elementtvpe\* 171 pointer-from-address function 358 print-action-lists function 199 print-actions function 199 \*print-binding-frames\* variable 80 \*print-catch-frames\* variable 81 \*print-command\* variable 200 \*print-handler-frames\* variable 82 \*print-open-frames\* variable 84 print-profile-list function 118 print-query function 303 \*print-restart-frames\* variable 84 process-alive-p function 237

process-allow-scheduling function 238 process-arrest-reasons function 238 process-break function 239 \*processes\* special variable 237 process-foreign-file function 153 process-foreign-file macro 152 \*process-initial-bindings\* special variable 239 process-interrupt function 239 process-kill function 240 process-lock function 240 process-name function 241 process-p function 242 process-plist function 242 process-priority function 243 process-reset function 243 process-run-function function 244 process-run-reasons function 244 process-send function 245 process-unlock function 246 process-wait function 246 process-wait-function function 247 process-wait-with-timeout function 248 proclaim function 55 profile macro 120 \*profiler-threshold\* variable 122 \*profile-symbol-list\* variable 122 ps function 248

## Q

query function 305

#### R

read-color-db function 26 read-foreign-modules function 157 rebinding macro 200 release-lock function 249 remove-advice function 201 removef macro 202 remove-special-free-action function 123 remove-symbol-profiler function 123 reset-profiler function 124 restore-sql-reader-syntax-state function 305 rollback function 306 room function 56

#### S

save-image function 125

save-tags-database function 88 sbchar function 203 schedule-timer function 249 schedule-timer-milliseconds function 251 schedule-timer-relative function 252 schedule-timer-relative-milliseconds function 254 select function 307 set-default-character-element-type function 204 set-default-generation function 128 setf timer-name function 255 set-gc-parameters function 129 set-minimum-free-space function 131 set-profiler-threshold function 132 set-promotion-count function 132 set-up-profiler function 133 \*sg-default-generation\* special variable 359 \*sg-default-size\* variable 359 simple-augmented-string type 360 simple-augmented-string-p function 360 simple-base-string type 59 simple-base-string-p function 204 simple-char type 205 simple-char-p function 205 simple-process-p function 256 simple-text-string type 206 simple-text-string-p function 206 socket-stream class 33 source-debugging-on-p function 134 special forms declare 42 special variables \*current-process\* 222 \*initial-processes\* 225 \*processes\* 237 \*process-initial-bindings\* 239 \*sg-default-generation\* 359 sql function 309 sql-expression function 309, 311 sql-operator function 312 sql-recording-p function 312 sgl-stream function 313 \*stack-overflow-behaviour\* variable 361 start-dde-server function 403 start-sql-recording function 314 start-up-server function 31 staticp function 361

status function 315 stchar function 207 step macro 59 \*step-compiled\* variable 62 \*step-filter\* variable 62 \*step-print-env\* variable 62 stop-sql-recording function 316 stream generic function 337 stream-advance-to-column generic function 327 stream-clear-input generic function 328 stream-clear-output generic function 328 stream-element-type generic function 63 stream-finish-output generic function 329 stream-force-output generic function 329 stream-fresh-line generic function 330 stream-line-column generic function 330 stream-listen generic function 331 stream-output-width generic function 331 stream-peek-char generic function 332 stream-read-byte generic function 333 stream-read-char generic function 333 stream-read-char-no-hang generic function 334 stream-read-line generic function 335 stream-read-sequence generic function 335 stream-start-line-p generic function 336 stream-unread-char generic function 338 stream-write-byte generic function 338 stream-write-char generic function 339 stream-write-sequence generic function 340 stream-write-string generic function 341 string type 63 string-append function 207 switch-static-allocation function 135 \*symbol-alloc-gen-num\* variable 135 system load 193.196

#### Т

text-string type 208 text-string-p function 208 time macro 64 timer-expired-p function 256 timer-name function 257 toggle-source-debugging function 136 top-level-form macro 209 total-allocation function 137 trace macro 65 \*traced-arglist\* variable 138 \*traced-results\* variable 66, 139 \*trace-indent-width\* variable 140 \*trace-level\* variable 141 trace-new-instances-on-access function 3 trace-on-access function 4 \*trace-output\* variable 71 \*trace-print-circle\* variable 141 \*trace-print-length\* variable 142 \*trace-print-level\* variable 143 \*trace-print-pretty\* variable 144 true function 210 types 16-bit-string 160 8-bit-string 159 augmented-string 343 base-char 160 base-character 161 base-string 35 character 36 extended-char 184 extended-character 184 simple-augmented-string 360 simple-base-string 59 simple-char 205 simple-text-string 206 string 63 text-string 208

#### U

unbreak-new-instances-on-access function 5 unbreak-on-access function 6 undefine-action macro 210 undefine-action-list macro 211 unnotice-fd function 258 \*unresolved-messages\* 158 unschedule-timer function 259 untrace macro 71 untrace-new-instances-on-access function 6 untrace-on-access function 7 update-record-from-slot generic function 318 update-record-from-slots generic function 319 update-records function 316 update-records-from-instance generic function 317

### V

valid-external-format-p function 93 variables \*active-finders\* 87 \*binary-filetype\* 106,147 \*binary-file-types\* 40, 106, 147 \*color-database\* 13 \*connect-if-exists\* 273 \*debug-print-length\* 77 \*debug-print-level\* 78 \*default-action-list-sorttime\* 171 \*default-database\* 279 \*default-database-type\* 279 \*default-libraries\* 157 \*default-package-use-list\* 102 \*default-process-priority\* 222 \*default-profiler-limit\* 103 \*default-simple-processpriority\* 222 \*extended-spaces\* 213,348 \*external-formats\* 185 \*file-encoding-detectionalgorithm\* 349 \*file-eol-style-detectionalgorithm\* 351 \*filename-pattern-encodingmatches\* 351 \*handle-existing-action-inaction-list\* 191 \*handle-existing-action-list\* 191 \*handle-existing-defpackage\* 115 \*handle-missing-action-in-actionlist\* 192 \*handle-missing-action-list\* 192 \*handle-warn-on-redefinition\* 193 \*hidden-packages\* 79 \*initialized-database-types\* 295 \*line-arguments-list\* 355 \*lispworks-directory\* 194 \*max-trace-indent\* 116 \*multibyte-code-page-ef\* 363 \*packages-for-warn-onredefinition\* 118 \*print-binding-frames\* 80 \*print-catch-frames\* 81

```
*print-command* 200
*print-handler-frames* 82
*print-open-frames* 84
*print-restart-frames* 84
*profiler-threshold* 122
*profile-symbol-list* 122
*sg-default-size* 359
*stack-overflow-behaviour* 361
*step-compiled* 62
*step-filter* 62
*step-print-env* 62
*symbol-alloc-gen-num* 135
*traced-arglist* 138
*traced-results* 66,139
*trace-indent-width* 140
*trace-level* 141
*trace-output* 71
*trace-print-circle* 141
*trace-print-length* 142
*trace-print-level* 143
*trace-print-pretty* 144
```

#### W

#### Y

yield function 262