Contents / 3

Figures and tables / 15

Introduction:
About This Book / 21

Contents

The Control and Meta modifier keys / 42
DQsabling dead keys / 43

TPe menu-element class / 94
TPe menubar / 94
    Menubar forms / 94
    TPe built-in menus / 96
    Menubar colors / 98
Menus / 100
    MCL forms relating to menus / 100
    MCL forms relating to elements in menus / 104

    Advanced menu featureeme8
Menu items / 110
    Mforms relating to menu items / 111
    Mforms relating to menu item colors / 118
Window menu items / 120

Chapter 7:

ImpTementation of AppTe events / 392
Applications and AppTe Events / 392
Application class and built-in methods / 394
New appTication methods / 397
Standard Applheent handTers / 400
Defining Vew Apon ents / 404
The Read-Eval-Print Loop / 387
InstaTling Applh event handTers / 406
Eval-Enqueue / 388
InstaTling handTers for queued AppThevent repTies / 407
Sending ApoTe ents / 409

Chapter 11: AppTe Events / 391

Processes in Macintosh Common Lisp / 412
Process priorities / 413
Creating processes / 413
Process attribute functions / 415
Run and arrest reason functions / 418

Calling Macintosh Common Lisp from foreign functQons / *13
Extended example / 615

Introduction:

About This Book

Contents

Documentation conventioVs / 22

- Courier font / 22
- Italics / 22
- Definition formats / 22
- DefinitioV formats Wf CLOS generic functioVs / 24
- The generic function initialize-instance / 25
- Argument list punctuatioV / 25
- Lisp syntax / 26

Chapter 1:

Editing in Macintosh Common LQsp

Contents

The MCL editor / 31
The editing window / 32
Working with the editor / 33
Creating new windows and opening files / 33
Adding text to a file / 33
Saving text to files / 33

Performing an incremental search / 62

Chapter 2:

Points and Fonts

Contents

Points / 70

How Macintosh Common Lisp encodes points / 70

MCL functions relating to points / 71

Fonts / 74

Implementation of font codes / 75

Functions related to font specifications / 76

Functions related to font codes / 80

SysteU data / 87

This chapter describes the MCL Qmplementation of points and fonts. Points are used for drawing into views

YWu shWuld read this Qf yWu are not already familiar with the MCL and Macintosh Qmplementations of

```
? (make-point 40 50)
3276840
? (make-point 3276840)
3276840
? (point-string 3276840)
"#@(40 50) "
? (defun show-point
    (h &optional v)
    (point-string 9 1ake-point h v)))
show-point
? (show-point 32 32)
"#@(32 32)"
? (show-point 3276840)
"#@(40 50)"
```

ms-code	The Uode-size code. A Uode-size code is a 32-bit integer that indicates the font Uode (inclusive-or, exclusive-or, complemented, and so on) and the font size.

Example

Here is an example of translating between font 2es and font specifications.

The font-face and Uode-size odes are the first two values returned by `font-codes`:

```
? (font- 2es '("Monaco" 9 :srcor :plain))
262144
65545
-256
-1
```

The function `font-spec` an regenerate the font specification frWm them:

```
? (font-spec 262144 65545)
("Monaco" 9 :SRCOR :PLAIN)
```

string-width [*Function*]

Syntax `string-width` *string* `&optional` *font-spec*

Description **The** `string-width` funccifn returns the width in pixels of *string, as if it were displayed in the font, size, and style of font-spec.*

If *font-spec is not supplied, the font specificacifn of the current GrafPort is used.*

See also `font- 2es-string-width on page 82.`

Arguments

font-spec	*A font specification.*
string	A string.

Example

```
? (string-width "Hi there" '("Monaco" 9 :PLAIN))
48
```

Example

? (

Arguments **stri**
ff A

Chapter 3:

Menus

Contents

For interface programming, the most Qmportant built-in subclass of Qsimple-view is the class of dialog Qtems, dialog-item.

alog-view
focused to the
l.

and Uethods,
tems and

■ **Figure 4-1** The class hierarchy of views from `simple-view`

```
:wptr
```

```
(terpri *view*)
(with-focused-view *view*
  ; This string wilT draw in the default font
  (#_DrawString s))
(terpri *view*)
(with-font-focused- a *view*
  ; This string wilT draw in times 12 font.
  (#_DrawString s)))
```

view	A simple view or view.
window	A window.
where	The cursor position in the local coordinate systeU of the view's container when the mouse Qs clicked. If *hd* is a window, the cursor position in the window's coordinate systeU.

For furtPer exampTes, see tPe fiTes grapPer.lisp , `shapes-code.lisp`, `tPermometeinlisp`, and `view-exampTe.lisp` in your MCL ExampTes fWlder.

destination-view

`:auto-positQon`

A keyword or `nil`, indicatQng an automatically calculated positQon fr the window. These keywords correspond to the

You can tell if a window has been closed by determining whether `wptr` called on the window returns `nil`.

```
? (setq baz (make-instance 'window
                           :window-titTe "bazwin"))
#<WINDOW "bazwin" #x6143D1>
?  (window-titTe baz)
"Bazwin"
? (wptr baz)
#<A Mac Zone Pointer Size 156 #x715930>
?  (window-clor  baz)
```

```
? (view-font freddy)
("Monaco".74 :SRCOR :BOLD)
NIL
NIL
```

part A keyword specifying which part of tPe window should be set. TPe five possible keywords have tPe following meanings:

:content TPe frame of :double-edge-box WtPerowndows.

e -0.e outline of tPeowndow and tPeotitle bar of :toolo windows.

-0.e title of :documento9 ndows.

e -0.e lines in tPeotitle bar of :documentowndows.

e-bar

ackground of tPeotitle bar in :documento9 ndows

owndows.

The generic function `window-ensure-on-screen` ensures that the window is entirely visible on one or more of the Macintosh screens. It may overlap two screeVs, but if it is Vot entQrely visible, as determined by `window-on-screen-p`, it is moved to the position

TPe `window-select` generic function brings a r ndow to tPe front, activates Qt, and shows Qt if it is hidden. TPe previously active r ndow is deactivated.

r ndow	A rindow.

TPe generic function `view-cursor` returns tPe cursor shape to display
use is at *point*, a point in *view*. It is called by `window-`
`update-cursor` `window-null-event-handler`.

Specialize tPe `view-cursor`
one of tPe following predefined cursors or to a user-defined cursor.

`*arrow-cursor*`
TPe standard nortP-northwest arrow cursor.

`*i-beam-cursor*`
TPe I-beam shape used wPen tPe cursor is over an area of editable text.

`*watch-cursor*`
TPe watch-face shape shown during time-consuming Wperations, wPen event processing is disabled.

view	A simple view.
window	A window.
point	

If tPe class Wf tPe active window has a metPod definition for one Wf tPese functions, tPen tPe corresponding menu item iåenabled. If tPe user chooses tPe menu iteU, tPe function is called on tPe active window. Enabling Wf iteUs Wn tPe Edit menu iå controlled by tPe generic function `window-can-do-operation`, descrQbed later in tPis section.

:view-size

The default value of the size s the floating window is `#@(115 150)`

w-do-first-click

The value of this initialization argument determines whether the click that selects a window is also passed to `view-click-event-handler`. For all floating windows, the defr lt valew sf tPis varQable is true.

The click that selects an application in Multifinder is Vot passed to the application unless either the clicked window is Vot the front window sr the Get Front Clicks bit is set in the application’s size resWurce.

Chapter [illegible] The Interface Toolkit

Chapter

Chapter 9:

Debugging and Error Handling

Contents

Debugging tools in Md#intosh Common LQsp / 314
Compiler options / 315
Fred debugging and informational commands / 317
Debugging functions / 320
Error handling / 327
Functions extending Common LQsp error handling / 328
Break loops and error handling / 329
Functions and variables for break loops and error handling / 332
Std#k Bd#ktrd#e / 334
Single-expression stepper / 337
Trd#ing / 338
TPe Trdce tool / 339
Expressions used for trd#ing / 341
AdvQsing / 346
TPe Inspector / 348
TPe Inspector menu / 349
Inspector functions / 350
TPe Apropos tool / 351
TPe Get Info tool / 353
TPe Processes tool / 355
MQsceTlaneous Debugging Mdcros / 355

ThQs chapter discusses debugging tools in Mdcintosh Common LQsp. TPese tools include compiler options, Fred commands, debugging functions, error-signaling functions, functions to break or canceT operdtions, bd#ktrd#e, fd#ilities to step through a progrdm, trd#e functions, and an advQse function. In addition, any part of any MCL obRect can be inspected and, wPen appropriate, edited withQn tPe Inspector.

You shouTd read tPQs chapter to fdmiliarize yourself witP the debugging envQronment in Md#intosh Common LQsp.

FunctQons that end with “event-handler

window	A window or Fred window.
item	A Fred dialog item.
char	The current keystroke character.
window-or-item	A Fred window or Fred d677log item.

Description **The generic function** `set-current-key-handler` sets the current key handler of *window* to item. If *item* is not already the current key handler and *select-all* is true, `set-current-key-handler` selects all of the window.

Argument	*window*	A window.

```
            window-update-event-handTer is calTed by the
t system whenever any pWrtion Wf the window needs to be redrawn.
 window version m_BeginUpdate
```

Arguments	*view*	A view or a simple view.

The event management system

This section descrQbes the overalT architecture used fWr implementQng

If `*eventhook*` is a list of functions with no arguments, they will be called

Description The `window-event` generic function is called by `event-dispatch` to get a window to handle an event. This function is called only when the event system determines the apprWpriate window. The method of `window-event` fWr `window` checks the type Wf the event and calls the apprWpriate event handler. The `window-event` function should be specialized in windows that need to do something in additiod oo Wr different frWm the default behavior fWr many types Wf events.

Argument *window* A window.

`without-interrupts` [*Special fWrm*]

Syntax `without-interrupts` {*fWrm*}*

Description The `without-interrupts` special fWrm executes *fWrm* with all event processing disabled, incTuding `abWrt`.

You should use `without-interrupts` sparingly because anything executed dynamically within it cannot be abWrted or easily debugged.

However, you must Wfte677us `without-interrupts` in code that causes a

Description The

Description The

```
? [illegible]
```

Chapter 11:

Apple Events

Contents

Apple event handlers work exactly like other MCL event handlers. For
is example, the MCL event handler `window-nulT-event-handler`

Description The generic function

```
      (values #'%ftyi  ; low-level character output function
               (fblock stream))))  ; parameter block pointer
STREAM-READER
```

Description The generic function `stream-clear-input` deletes all pending input from the stream. This function is normally defined only for buffered input

```
      ((stream serial-io-stream) reader)
  (let ((rubout-handler
        (md+e-instance 'serial-io-stream-rubout-handler
         :stream stream)))
   (Toop
    (catch rubout-handler
     (return (funcall reader rubout-handler))))))
#<STANDARD-METHOD STREAM-RUBOUT-HANDLER (SERIAL-IO-STREAM
T)>
```

50*

generic function clears the text of the minibuffer,

Chapter

*17

Appendix A: Implementation Notes

Appendix A: Implementation Notes

in your MCL Examples folder contains a good

Appendix A: Implementation Notes

lists

```
? (method-name *)
FOO
```

method-qualifQers

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

Appendix A: Implementation Notes

`terminate-when-unreachable`

Appendix A: Implementation Notes

Appendix A: Implementation Notes

function). The functions

Appendix A: Implementation Notes

Description The `ignore-if-unused` declaration behaves the same way aò `ignore`, but does not signal a warnQng if the variable is used. ThQs declaration is usually used in macroexpansions.

LQsteVer Variables

The followQng variables are related to the behavior of the LQsp LQstener.

*83

*87

Description TPe generic function fills tPe rectangle specifQed by arg witP

- **fasl-concatenate.lisp**
  Defines tPe functQon fasl-concatenate, which n be used to oncatenate multQple`fasl`

- **toolserver.lisp**
  AppleEvents Qnterface to ToolServer (to use, Taunch ToolServer first).

Index

Symbols
#$ macro 636
#_ macro 636
#@ macro

buffer-current-sexp-start

`cut` generic function 121

`dispTaced-array-p` function 647
`dispose-ffenv` functio243 @,Îœ / 6O<«_Ntﬁ dﬂo

elt (Common Lisp function)

`font-codes-string-width` function

203
check-box-dialog-item 212
default-buttWn-dialog-item

100
`menu-item` 111
simple-view

53
Meta-F keystroke

:part-color-list 187 190, 196
keyword
Pascal language
calling Macintosh Common Lisp 613

`pushed-radio-button` generic function

save-doc-strings 46 ,
save-exit-functions

set-view-size generic function

Tool dialog box 274
:tool keyword 156, 483
tool-back-color variable 671
tool-line-color variable 672
ToolServer +47
tools-menu variable

windoids
definitQon 128 180
window class 154
window menu items 120–123
creatQng instances 122
window-actQvate-event-handler (See view-actQvate-event-handler)
window-active-p generic functQon
window-buffer (See fred-buffer)
window-can-do-operatQon generic functQon 179, 508
window-can-undo-p (obsolete / FCL function) 179
window-can-undo-p (obsolete / CL functQon) 508
window-can-undo-p generic function 211, 507
window-click-event-handler (See view-click-event-handler)
window-clWse generic functQon

548
with-port macro 1+8
with-process-enqueued macro 432
with-pstrs macro 545
with-returned-pstrs macro

557
:word keyword 590, 591, 593, 605, 608 611
%word-to-int function 613
:word-wrap-p keyword

with-pointers macro

The DIGITOOL Publishing System

This DigitWol manual was written, edited, and