New Technical Notes
Macintosh

®

Developer Support

Macintosh Allegro Common Lisp Features
Platforms & Tools **M.PT.CommonLisp**

Revised by: Paul Snively February 1990
Written by: Guillermo Ortiz April 1989

This Technical Note describes some known problems and provides solutions to these problems for the Macintosh Allegro Common Lisp™ package which is available from Apple Computer, Inc. You should note, however, that although Apple acquired Coral Software and is selling Macintosh Allegro Common Lisp, Apple is not currently distributing any other products which had been developed or previously sold by Coral Software.
Changes since April 1989: Noted 1.3.1 documentation errors, corrected erroneous floating-point patch for version 1.2.2, updated 1.2.2 information which is not relevant to 1.3.1, corrected APDA part number, added examples of high-level printing functions in 1.3.1, an array-dialog-item example, and information explaining how to get the Victoria-Day release of Portable Common LOOPS (PCL) to compile correctly under 1.3.1.

The current supported version of the Macintosh Allegro Common Lisp package (MACL) is 1.3.1; if you have an earlier version of this product, you should obtain an upgrade through APDA (part number M0229LL/C), as Apple only supports the current version.

Some Known Problems and Solutions

- **Typographical errors in the 1.3.1 documentation.**
  The following symbols, documented in the "Menus" chapter, are missing a hyphen (-) in the documentation. Their proper names are as follows:

```
add-menu-items              menu-item-enable
remove-menu-items           menu-item-enabled-p
menu-items                  menu-item-check-mark
*menu-item*                 set-menu-item-check-mark
menu-item-title             menu-item-style
set-menu-item-title         set-menu-item-style
menu-item-disable           *window-menu-item*
```

  The following keywords are misspelled in the "Menus" chapter and do not appear in the index:

```
:menu-items                 :menu-item-colors
:default-menu-item-title    :menu-item-checked
:menu-item-title
```

  The following symbol is misspelled only in the index:

```
menu-update
```

The following symbol is misspelled only in its descriptive paragraph:

```
menu-item-update
```

- **Old versions crash on Macintoshes with a 68030 processor.**
  Versions of MACL prior to 1.2.2 may crash when running on 68030 machines because they are not compatible with this processor. You must upgrade to version 1.2.2 or later to solve this problem.

- **Some 1.2.2 packages do not run in the background.**
  Some MACL 1.2.2 packages which were distributed by APDA did not have the `'canBackground'` flag in the `'SIZE'` resource set, and will prevent background operation when running under MultiFinder. You can check for this problem by launching ResEdit and opening the `'SIZE' ID=-1` resource in MACL. If the `'canBackground'` flag is not set, you should set it.

- **Practice safe hex.**
  Due to its segment numbering scheme, MACL is very sensitive to viral infections. If things stop working for no apparent reason, check for viruses. You did back it up, didn't you?

- **Shut Down and Restart do not work in 1.2.2**
  Under MultiFinder, if you select Shut Down or Restart from the Special menu, the "going away" process stops with MACL 1.2.2 until you Quit it manually. This feature is fixed in 1.3.1.

- **Color dialogs and menus are not supported in 1.2.2.**
  MACL 1.2.2 does not support color dialogs, alerts, or menus. This feature is present in 1.3.1.

- **Only RAM pointers please.**
  Current versions of MACL cannot handle pointers into ROM or NuBus memory.

- **More memory?**
  Current versions of MACL are limited to supporting eight megabytes of memory. Future versions will support as much memory as the Macintosh OS supports.

- **Problems displaying PICTs on windows with 1.2.2.**
  There is a problem with clipping when displaying pictures that require resizing in 1.2.2. This problem has been fixed in 1.3.1, which utilizes a completely new view system modeled after MacApp. For developers still working with 1.2.2, the way to work around this problem is to replace the definition of `START-PICTURE` in the file QuickDraw.lisp with the following:

```
(defobfun (start-picture *window*) (&optional left top right bottom)
 (if (rref wptr window.picsave)
  (error "A picture may not be started for window: ~a.
          since one is already started" (self)))
(unless left (setq left (rref wptr window.portrect)))
(with-rectangle-arg (r left top right bottom)
 (with-port wptr
   (_cliprect :ptr r)
   (have 'my-hPic (_OpenPicture :ptr r :ptr))))
nil)
```

```
       ;;<this just adds a (_cliprect :ptr r) to the old definition>
```

- **1.2.2 crashes on Macintoshes with a 68882 coprocessor.**
  Some old versions of MACL can crash on Macintoshes with a 68882 floating point coprocessor. Certain errors, such as a floating point divide-by-zero, are not caught and crash the machine instead of being reported as Lisp errors. This bug is fixed in MACL 1.3.1, but developers still using versions prior to 1.3.1 can include the following patch in the file init.Lisp, so it gets executed before anything else:

```
(in-package "CCL")

(defun validate-fp-handler (handler)
 "If the HANDLER argument appears valid, return it.  Otherwise,
  make a new one which Does The Right Thing.
  Note that the kernel will restore system floating-point
  exception handlers on exit, so we don't worry about that here."
 (let* ((old-addr (%ptr-to-int handler))
        (words (list
                #o026417 ; move.l sp,-(a6)
                #o171447 ; fsave -(sp)
                #o027400 ; move.l d0,-(sp)
                #o027401 ; move.l d1,-(sp)
                #o070000 ; moveq #0,d0
                #o010057 ; move.b 9(sp),d0
                #o000011
                #o004367 ; bset #27,8(sp,d0.w)
                #o000033
                #o000010
                #o171000 ; fmove.l fpsr,d0
                #o124000
                #o031074 ; move.w #$3400,d1
                #o032000
                #o141100 ; and.w d0,d1
                #o063012 ; bne.s @1
                #o021037 ; move.l (sp)+,d1
                #o020037 ; move.l (sp)+,d0
                #o171537 ; frestore (sp)+
                #o054116 ; addq #4,a6
                #o047163 ; rte
                #o027136 ; @1: move.l (a6)+,sp
                #o041247 ; clr.l -(sp)
                #o171537 ; frestore (sp)+
                #o171000 ; fmove.l d0,fpsr
                #o104000
                #o047371 ; jmp <old-handler>
                (logand (ash old-addr -16) #xff)
                (logand old-addr #xffff)))
        (len (length words)))
   (unless (= (%get-signed-word handler)
              (car words))
     (setq handler (%register-trap #xa11e 384 (* 2 len)))  ;(_Newptr:d0 (* 2 len):a0)
     (dotimes (i len)
       (%put-word handler (pop words) (+ i i))))
   handler))

(defun ccl-using-fpu-p ()
  (not (= 0 (%get-byte (%get-ptr (%int-to-ptr #x904)) #x-130))))

(when (ccl-using-fpu-p)
 (let* ((addr (%int-to-ptr #xc8)))
   (%put-ptr addr (validate-fp-handler (%get-ptr addr))))
 t)
```

- **High-level printing functions.**
  The high-level printing functions that were available in 1.2.2 are no longer available in 1.3.1. Following is the code necessary to implement hardcopy for `*window*` objects:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;hardcopy.lisp
;;
;;
;;Copyright 1988-99 Apple Computer, Inc.  All Rights Reserved.
;;
;; defines a very basic printing routine for windows
;;
;; This code sets the window's wptr to a printer grafport
;; and then calls view-draw-contents
;;
;; This code does allow printing of Dialogs
;;


(eval-when (eval compile)
 (require 'traps)
 (require 'records)
 (defconstant $PrintErr #x944)
 (defconstant $prJob.bJDocLoop (+ 62 6))
 (defconstant $iPrStatSize 26)
 (defconstant $bSpoolLoop 1)
 (defconstant $err-printer 94)
 (defconstant $err-printer-load 95)
 (defconstant $err-printer-start 97)
)

(defun prchk (&optional (errnum $err-printer)
                       &aux (print-error (%get-signed-word $PrintErr)))
  (unless (zerop print-error)
     (ccl::%signal-error errnum print-error)))

(defobfun (set-view-wptr *view*) (new-wptr)
 (setf (objvar wptr) new-wptr)
 (let ((subviews (objvar view-subviews)))
   (dotimes (index (length subviews))
     (ask (aref subviews index)
       (set-view-wptr new-wptr)))))

(defobfun (window-hardcopy *window*) ()
 (window-select)
 (unwind-protect
   (with-cursor *arrow-cursor*
     (_PrOpen)
     (prchk $err-printer-load)
     (let ((pRec (get-print-record)))
       (when (_PrJobDialog :ptr pRec :boolean)
         (let ((*hc-page-open-p* nil) (ccl::*inhibit-error* t) err)
           ;_PrOpenDoc puts up a dialog window which causes the event system
           ;to get confused.  So we do the whole thing without interrupts, and
           ;make sure to clean up before handling errors.
           (declare (special *hc-page-open-p* ccl::*inhibit-error*))
           (setq err (catch-error-quietly
                       (without-interrupts
                        (with-port (_PrOpenDoc :ptr pRec :long 0 :long 0 :ptr)
                          (let ((window-ptr wptr)
                                (hardcopy-ptr (ccl::%getport)))
                            (unwind-protect
                              (with-dereferenced-handles ((ppRec pRec))
                                pprec
                                (prchk $err-printer-start)
                                (unwind-protect
                                  (progn
                                    (set-view-wptr hardcopy-ptr)
                                    (_PrOpenPage :ptr hardcopy-ptr :long 0)
                                    (view-draw-contents)
                                    (_PrClosePage :ptr hardcopy-ptr))
                                  (set-view-wptr window-ptr)))
                              (_PrCloseDoc :ptr hardcopy-ptr))))
                        (when (eq (%hget-byte pRec $prJob.bJDocLoop)
                                  $bSpoolLoop)
                          (prchk)
                          (%stack-block ((StRec $iPrStatSize))
                           (_PrPicFile :ptr pRec :long 0 :long 0 :long 0 :ptr StRec))
                          (prchk)))))
           t))))
   (_PrClose)))

;;unfortunately, this doesn't work for dialogs
(defobfun (window-hardcopy *dialog*) ()
 (message-dialog "Cannot print of dialogs at this time"))

#|
(require 'quickdraw)
(setq foo (oneof *window*))

(defobfun (view-draw-contents foo) ()
 (frame-rect 10 10 100 100)
 (usual-view-draw-contents))

(setq bar (oneof *view*
                :view-container foo
                :view-position #@(150 150)))

(defobfun (view-draw-contents bar) ()
 (paint-oval 10 10 100 100)
 (usual-view-draw-contents))

(ask foo (window-hardcopy))
|#
```

- **I want to use a grapher with MACL.**
  A simple grapher is included as an example source file with 1.3.1.

- **Missing array-dialog-item example in 1.3.1 Examples folder.**
  The 1.3.1 documentation mentions an array-dialog-item example, but it is missing from the Examples folder on the disk. Following is the missing example:

```
;;;;;;;;;;;;;;;;;;;;
;;
;;  array-dialog-items
;;
;;  ©1989, Apple Computer, Inc
;;
;;
;;  a subclass of table-dialog-items used to display two-dimensional arrays
```

```
;;
(in-package :ccl)

(export '(*array-dialog-item* table-array set-table-array) 'ccl)

(defobject *array-dialog-item* *table-dialog-item*)

(defobfun (exist *array-dialog-item*) (init-list)
 (let* ((the-array (getf init-list :table-array (make-array '(0 0))))
        (dims (array-dimensions the-array)))
   (unless (eq (length dims) 2)
     (error "table-array ~s is not of rank ~d" the-array 2.))
   (have 'my-array the-array)
   (usual-exist (init-list-default
                 init-list
                 :table-dimensions (make-point (car dims) (cadr dims))))))

(defobfun (cell-contents *array-dialog-item*) (h &optional v)
 (unless v
   (setq v (point-v h)
         h (point-h h)))
 (aref (objvar my-array) h v))

(defobfun (table-array *array-dialog-item*) ()
 (objvar my-array))

(defobfun (set-table-array *array-dialog-item*) (new-array)
 (setf (objvar my-array) new-array)
 (inval-dialog-item))


#|
(setq table (oneof *array-dialog-item*
                  :table-array #2a((a1 b1 c1)
                                   (a2 b2 c2)
                                   (a3 b3 c3))))
(oneof *dialog*
      :dialog-items (list table))

(ask table (set-table-array #2a((x1 y1 z1)
                                (x2 y2 z2)
                                (x3 y3 z3))))

|#
```

- **Common Lisp Object System (CLOS).**
  Future versions of MACL will support an Apple implementation of CLOS, but you can use PCL, a portable implementation of CLOS, until that time. PCL is available from various sources, including APDA.

  If you have the Victoria-Day release of PCL, the following changes to the source code allow it to compile successfully under MACL 1.3.1:

  In file defsys.lisp:

  Find the `defvar` for `*pcl-directory*`. Within it, find the conditional for MACL (`#+:coral`). Change the pathname parameter to point to your PCL folder (e.g., `"ccl;PCL:"`). Also find the `let` of `files-renamed` and change its binding to `nil`.

In file coral-low.lisp:

Comment out both the `ccl::add-transform` and its `inline` proclamation. Neither is helpful in 1.3.1 (in fact, the `add-transform` is broken with respect to 1.3.1).

In file fin.lisp:

Immediately before the closing "`); End of #+:coral`" that you find near the end of the file, add:

```
(defun print-uvector-object (obj stream &optional print-level)
  (declare (ignore print-level))
  (print-object obj stream))

(pushnew (cons 'ccl::funcallable-instance #'print-uvector-object)
         ccl:*write-uvector-alist* :test #'equal)
```

In addition to these code changes, there are some environmental settings that are useful or necessary when compiling Victoria-Day PCL. You should use the following settings:

```
(setq *WARN-IF-REDEFINE-KERNEL* nil)
(setq *COMPILER-WARNINGS* nil)
(setq *FASL-COMPILER-WARNINGS* nil)
(setq *FAST-EVAL* nil)
```

These settings eliminate loads of warnings that you would otherwise get when compiling or loading PCL. In particular, you must bind or assign `nil` to `*FAST-EVAL*` for the file test.lisp to load correctly.

Allegro Common Lisp is a trademark of Franz Inc.