Macintosh Common Lisp 2.0
Release Notes

These Release Notes supplement the material in *Getting Started with Macintosh Common Lisp 2.0.*

MCL 2.0 conforms to the Common Lisp standard as it is described in Guy L. Steele's *Common Lisp: The Language (Second Edition)*. The ANSI Common Lisp standard continues to evolve through the work of the ANSI X3J13 subcommittee. In a few cases we have incorporated post-*Steele* details in MCL 2.0, based on our understanding of the ongoing work of the subcommittee. The second edition of *Steele* (which we'll refer to as *Steele2* below) is, however, the Common Lisp reference intended to be used with this implementation.

Previous versions of our Lisp product have been called "Macintosh Allegro Common Lisp" — this is the "MACL" you will encounter as you read this document. MACL 1.x versions were based on the first edition of Steele's book (*Steele1* from now on). Actually, the lineage of this Lisp goes back farther still, to a product called "Coral Common Lisp" (this is why, for example, the package in which we have defined our extensions to the Common Lisp standard is called "CCL").

This document contains five sections:

Part 1: Installing Macintosh Common Lisp 2.0 2
Part 2: General Notes on MCL 2.0 4
Part 3: Third-Party MCL-Based Tools 12
Part 4: Upgrading from MACL 1.x to MCL 2.0 14
Part 5: Upgrading from MCL 2.0b1 to MCL 2.0 18

If you are just starting out with MCL 2.0, you probably do not need to read past the first section for now. After you've gone through *Getting Started* and become familiar with the MCL environment, come back and skim Parts 2 and 3; you may find the information they contain useful.

Although some effort has been made to divide information among the final two sections in a reasonable way, you may find it useful to skim both of them.

Part 1— Installing Macintosh Common Lisp 2.0

If you have a CD-ROM drive attached to your Macintosh, simply drag the "MCL 2.0" folder from the *MCL 2.0 CD-ROM* to your hard disk, then proceed to step 9 below.

Installing from Floppy Disks

The three MCL 2.0 floppy disks each contain one segment of a Compact Pro archive. The archive is "self-extracting": you need only insert disk #1, double-click on the "MCL 2.0.sea" application's icon, then follow Compact Pro's instructions. Compact Pro will create a folder called "MCL 2.0" containing the entire MCL 2.0 environment.

Make sure you have 5.5 megabytes available on your hard disk, then follow these steps:

1. It is a good idea to back up the three disks shipped with MCL 2.0, just in case you need to reinstall someday and something has happened to the originals.
2. Insert disk #1.
3. Double-click on the "MCL 2.0.sea" application.

After a brief pause, Compact Pro will ask you to show it the "Final Segment" of the archive. When it does:

4. Eject disk #1.
5. Insert disk #3.
6. Select "MCL 2.0.sea.3" and click on "Load".

Next Compact Pro will ask where you want to put the "MCL 2.0" folder.

7. Select an appropriate location on your hard disk for Compact Pro to put the "MCL 2.0" folder, then click on "Extract".
8. Insert each disk as Compact Pro requests it.

After all the material has been extracted from the archive, your “MCL 2.0” folder will contain the following:

Examples folder
Contains various examples of MCL 2.0 code and useful utilities. These files are described briefly later in this document.

Interface Tools folder
Contains a system that helps you design dialogs interactively. Instructions for its use are in the file **About Interface Tools**.

Library folder
Contains subsystems that can be loaded when required (the interface to QuickDraw or to the Foreign Function Interface, for example).

MCL 2.0
This is the MCL 2.0 application.

MCL Help
MCL Help Map.Fasl
These two files contain information needed by the on-line help and documentation mechanism.

PTable
This file contains a system “extension” (an “INIT” for those of you using System 6) to improve the performance of garbage collection if you are not using Virtual Memory. Its use is discussed in the “General Notes on MCL 2.0” section of this document.

Series folder
Contains an implementation of the Series system described in Appendix A of *Steele2*. This implementation is by Richard Waters, the author of that appendix. The Massachusetts Institute of Technology holds the copyright on this material (please see the headers of the files in this folder for more details).

Now, to finish the installation process:

9. Move the “PTable” file from your MCL 2.0 folder to your System Folder (if you are using System 7, put it into the Extensions folder in your System Folder).

10. Restart your Macintosh so that the PTable extension can take effect.

Note:
Disk #3 also contains a Compact Pro utility called “Extractor”. Use this program if you want to get a few of the files out of the archive without extracting the entire environment.

Part 2— General Notes on MCL 2.0

This section contains notes on various details of the MCL 2.0 environment. It is organized roughly into four sections:

A few general notes;
Overview of what's in the Examples and Library folders;
Specific notes on some files in those folders;
Overview of what's on the CD-ROM.

You'll probably find this information more useful if you read it after you have some familiarity with the MCL environment.

Keeping in Touch

Chapter 5 in *Getting Started* discusses some ways to get in touch with us to report bugs (by sending electronic mail to bug-mcl@cambridge.apple.com) and to participate in electronic mail based discussions with your fellow users (a discussion group at info-mcl@cambridge.apple.com). In addition, if we produce "patches" to fix reported bugs and Tech Notes to discuss issues as they arise, they will be made available via electronic mail. All of these are available to anyone who can get access to the Internet (this includes those who can get access to the Internet through some intermediary service like AppleLink or Compuserve). To get details of this service, send email to the Internet address:

archive-server@cambridge.apple.com

The message should consist of the word HELP (this should be the body of the message, not the Subject line).

This archive will also contain at least some of the material contained on the CD that accompanies MCL 2.0 (for those of you without CD-ROM drives).

EGC and PTable

The Ephemeral Garbage Collector (EGC) makes use of your Macintosh's Memory Management Unit (MMU) if there is one, and the PTable extension supports the EGC in this by setting a reasonable memory page size (see "Memory Management Unit Support" in Appendix A of the *Reference Manual*). If you are using Virtual Memory, System 7 configures the page size for you, so you do not need the PTable extension. Similarly, you do not need PTable if your system does not have an MMU (as is the case with the Macintosh LC, for example) or if you are running AU/X. If PTable recognizes that it does not need to set the page size, its icon will appear with an "X" through it when you start up your system. In this event, or if you never intend to enable the EGC, you may remove PTable from your System folder.

If you would like to decide programmatically whether to enable the EGC (perhaps in an application delivered to a user whose Macintosh configuration you cannot predict), use the function EGC-MMU-SUPPORT-AVAILABLE-P; it will return a non-nil value if MCL believes the EGC can operate efficiently. This function does not appear in your documentation. Note that the decision to use the EGC is ultimately a subjective one — whatever this function returns, you may wish to give it a try, and see what you think.

Records

Describing the :include option of slot-descriptor in the defrecord macro, the *Reference Manual* says “In the first slot description, the value of :include is automatically true”. While this was the behavior of MCL 2.0b1, it is no longer — :include now defaults to nil for all slots.

Save-Application

SAVE-APPLICATION takes a new keyword argument — :MEMORY-OPTIONS. This argument gives you some control over how the saved application apportions and uses the memory available to it. The value of the :MEMORY-OPTIONS keyword argument is itself a list of keyword/value pairs, which are interpreted as follows:

Keyword	**Default value**
:MAC-HEAP-MINIMUM	102400
:MAC-HEAP-MAXIMUM	409600
:MAC-HEAP-PERCENTAGE	5
:STACK-MINIMUM	32768
:STACK-MAXIMUM	184320
:STACK-PERCENTAGE	6
:LOW-MEMORY-THRESHOLD	24576
:COPYING-GC-THRESHOLD	2147483648

Given an application partition size of *n* bytes, the MCL kernel reserves roughly

```
(max
        mac-heap-minimum
        (min mac-heap-maximum (* n (/ mac-heap-percentage 100)))))
```

bytes for the “mac heap”, and roughly

```
(max
        stack-heap-minimum
        (min stack-heap-maximum (* n (/ stack-heap-percentage 100)))))
```

bytes for MCL’s control and value stacks. (Each stack uses roughly half of this space.) The MCL kernel signals an “OUT-OF-MEMORY” condition when, after GC, free space in the lisp heap is less than the value of the :LOW-MEMORY-THRESHOLD option.

If free space in the lisp heap exceeds the value of the :COPYING-GC-THRESHOLD option when the application starts up, the MCL kernel arranges that a two-space copying GC algorithm is used for (non-ephemeral) garbage collection. This algorithm *may* improve GC performance in some large VM environments, at the cost of roughly halving the amount of memory available for the allocation of lisp objects.

PowerBook 170

Due to a bug in some versions of the PowerBook 170's system software (including 7.0.1), MCL 2.0 is unable to recognize certain floating point exceptions after the 170 has been in "Rest" mode. If you divide by zero in this circumstance, for example, MCL will simply return a very large number rather than signaling an error. If you notice this behavior on the 170 and find it to be a problem, consider disabling "Rest". This will, of course, drastically reduce the length of time your 170 will operate on a single charge of its battery. The PowerBook 100 and 140 do not have floating point hardware; they are not affected by this problem.

On Defsetf

The following note concerns a fairly esoteric aspect of Defsetf (feel free to skip over it if it doesn't make immediate sense).

Defsetf has been changed to match the behavior specified in *Steele2.* Certain idioms which appeared to "work" in previous versions of MCL — and which may also appear to "work" in other CL implementations — did so because the old implementation of DEFSETF incorrectly caused the expansion functions it generated to perform DEFMACRO-like destructuring over a list of GENSYM-ed temporary variable names.

This change may require changes to expansion functions which expect &REST parameters to be bound to a list of GENSYMs. For example:

```
(defsetf foo (&rest args) (new-value)          ; The old way
        `(set-foo ,new-value ,@args))
```

is no longer correct, because "ARGS" will be bound to a gensym (as opposed to a list of gensyms) at expansion time. This sort of expansion function must instead be written as:

```
(defsetf foo (&rest args) (new-value)          ; The new way
        `(apply #'set-foo ,new-value ,args))
```

Note that the "new way" will not work correctly in implementations that, like previous versions of MCL, misunderstood DEFSETF's specification.

What's in the Examples Folder

The Examples folder contains various MCL utilities. Each of the files contains comments that serve as brief documentation.

APPLEEVENT-TOOLKIT.LISP
: Provides useful functions for sending and processing AppleEvents.

ARRAY-DIALOG-ITEM.LISP
: A Table-Dialog-Item subclass for displaying arrays.

ASSORTED-FRED-COMMANDS.LISP
: FRED, MCL 2.0's editor, is fully programmable. This file contains examples of additional FRED commands.

AUTO-FILL.LISP
: A simple autofill mode for FRED.

BINHEX
: The two files in this folder contain an example of a "stand-alone" application: follow the instruction in Binhex.lisp to create an application in which the user need not know that he or she is using a Lisp-based system. The application encodes and decodes "BinHex" files.

BOYER-MOORE.LISP
: The Boyer/Moore string search algorithm. Replaces MCL's "Search Files" algorithm with the faster Boyer/Moore and gives feedback as the search progresses.

CHECK-AND-CHANGE.LISP
: This file contains code used in an example in Chapter 6 of the Reference Manual. The code illustrates the use of enter-key-handler and exit-key-handler.

COMPARE.LISP
: A compare utility, written by Laura Bagnall Linden of TERC.

DEFOBFUN-TO-DEFMETHOD.LISP
: This file automates conversion of simple Object Lisp programs (Object Lisp was MACL 1.x's object system) to CLOS (the Common Lisp Object System used by MCL 2.0). In a FRED window, it changes instances of (defobfun (function type) args body) to (defmethod function ((type type) args) body). Queries before each change.

DRIVER.LISP
: An example of how to access the Device Manager's drivers from Macintosh Common Lisp. Used by "serial-streams.lisp".

ESCAPE-KEY.LISP
: Makes the Escape key cause the next character to behave as if the Meta key were pressed.

EVAL-SERVER.LISP
: Handles eval, dosc, and scpt AppleEvents.

FF-EXAMPLES
: Three of the files in this folder ("ff-example.c", "ff-example.lisp", and "ff-example.test") contain code to demonstrate the use of MCL 2.0's Foreign Function interface. The fourth file ("ff-example.c.o") is the object file, compiled by the MPW C compiler, for "ff-example.c".

FONT-MENUS.LISP
: Defines a set of hierarchical menus that can be used for setting the font of the current window.

FRED-WORD-COMPLETION.LISP
: Provides word completion for symbols in a FRED window.

GRAPHER.LISP
: Implements the base functionality for node and grapher windows.

ICON-DIALOG-ITEM.LISP
A Dialog-Item subclass for displaying an icon.

MAC-FILE-IO.LISP
Implements something similar to the high-level file I/O primitives in *Inside Macintosh*. It does not support asynchronous I/O. Used by "boyer-moore.lisp" and "picture-files.lisp".

MARK-MENU.LISP
Adds a menu of marks, much like MPW's. Marks are not saved with the file.

MOUSE-COPY.LISP
When this file is loaded, command-click copies the expression nearest the cursor location to the location of the insertion point.

NOTINROM
This folder contains Michael Engber's package implementing most of the "traps" that are "Not in ROM". This package is discussed in Part 4 of this document, under "Traps".

OLD-DIALOG-HOOKS.LISP
Hooks to make the new dialog package more compatible with that of MACL 1.x. This code will run, but is intended principally as documentation of what has changed since that version.

PICT-SCRAP.LISP
This file defines a scrap-handler for scraps of type PICT. Once it is loaded, windows that copy and paste PICTs are able to share their work with other applications.

PICTURE-FILES.LISP
Examples of reading and writing picture files, adapted from the code on page V-88 of *Inside Macintosh.* Shows how to draw a PICT file in a window.

PRINT-CLASS-TREE.LISP
Code to print a simple class-tree.

QUERY-REPLACE.LISP
Implements a query-replace function in FRED, using Control-Meta-R.

SCROLLING-WINDOWS.LISP
Implements a new class of windows that contain scroll bars and a scrollable area.

SERIAL-STREAMS.LISP
Implements a class of serial streams, which inherit from drivers and provide a stream interface to the serial drivers on the Macintosh.

SHAPES-CODE.LISP
The Lisp version of a classic Mac programming exercise. Creates a window containing a drawing area and two buttons: "circles" and "squares." Clicking in the drawing area draws a circle or square; clicking either button redraws all the shapes.

TEXT-EDIT-DIALOG-ITEM.LISP
Implements text-edit-dialog-items. If FRED is too big for your application, you may wish to replace editable-text-dialog-items with text-edit-dialog-items.

THERMOMETER.LISP
A simple thermometer that displays one or more values in a rectangular area. GC-THERMOMETER and FILE-THERMOMETER are two examples provided in the file.

TOOLSERVER.LISP
AppleEvents interface to ToolServer (to use, launch ToolServer first). ToolServer is a stand-alone, tool-execution environment for Macintosh Programmer Workshop (MPW) tools such as the C and Pascal compilers and linker. ToolServer makes use of the AppleEvent feature of System 7. ToolServer is currently available on E.T.O. (Essentials - Tools - Objects), a CD-ROM subscription series distributed by Apple through APDA.

TURTLES.LISP
A simple object-oriented turtle graphics package.

UK-KEYBOARD.LISP
A FRED extension to make the sharp sign character ("#") easier to type on keyboards like those used in the United Kingdom.

VIEW-EXAMPLE.LISP
A simple example of views code.

WEBSTER.LISP
Defines Control-X $ to look up the definition of a word when you have access to a "Webster server".

WINDOID-KEY-EVENTS.LISP
How to make a windoid handle key events and null events.

What's in the Library Folder

Most of the files in the Library folder contain code that provides functionality that is not included in the MCL 2.0 image. If you need to use the interface to QuickDraw, for example, this form will load it:

```
(require :quickdraw)
```

Some of the files in the Library folder contain code that is autoloaded by MCL 2.0 when you try to use its functionality (the first time you type a form that uses LOOP, for example, "loop.lisp" is automatically loaded). These are files that will be autoloaded (if there is a compiled version of the file, it will be used instead): "help-manager.lisp", "lisp-package.lisp", "loop.lisp", and "resources.lisp".

A few of the Library files contain code that is already a part of the MCL 2.0 image (that is, you never need to load them). These files, provided for people who want to write extensions or who are just curious, are: the files in the Inspector folder and the files "scroll-bar-dialog-items.lisp" and "pop-up-menu.lisp".

Loop

The Library file "loop.lisp" is supplied in its source form only. As mentioned above, this file is loaded automatically the first time you use a loop construct. It will load much faster if you compile it (to "loop.fasl").

MCL Help and MCL Help-Map

You can modify the "mcl help" Library file to customize the documentation strings it contains. You must then create a new "map" using "make-help-map.lisp". This file contains instructions.

The Foreign Function Interface and Old Versions of MPW

The “ff-example.lisp” file (it’s in “examples:ff examples:”) contains the following:

```
(ff-load “ff;ff-example.c.o”
        :ffenv-name 'test
        :libraries '(“clib;StdCLib.o”
                “mpwlib;interface.o”))
```

The libraries referred to are as they exist in MPW 3.2 and 3.2.x. If you are using an earlier version of MPW, you will need to reference “clib;Cinterface.o” before “mpwlib;interface.o”.

MacTCP

The Library file “mactcp.lisp” mistakenly contains a call to the debugger. Please remove the line that says:

```
(dbg length)
```

Picture-Files

If you use the Examples file “picture-files.lisp” then do a (save-application ...), you will get an error when you start up the new application. This is because *STD-BITS-PROC* still contains an obsolete pointer (a dead-macptr). Change the (defvar *std-bits-proc* ...) in the file to the following:

```
(defvar *std-bits-proc* nil)
(def-load-pointers init-std-bits-proc ()
        (setq *std-bits-proc* (%int-to-ptr 0)))
```

What’s on the MCL 2.0 CD-ROM

MCL 2.0

This folder contains a complete installed copy of Macintosh Common Lisp 2.0, ready for use on a hard disk. To install MCL 2.0 on your hard disk, simply drag this folder onto your hard disk.

MCL 2.0 Floppy Disks

This folder contains four folders. Three of these folders contain the contents of the three disk Macintosh Common Lisp 2.0 floppy disk set. To re-create any of the MCL 2.0 disks, drag the contents of the respective folder onto an 800K (or larger) double-sided floppy disk. The disk must be newly formatted using System 7.0 or later. (Disks formatted with System 6 will not have enough available space.) Alternatively, you can use the “image” files in the “MCL 2.0 Images” folder with the DiskCopy utility (in “Developer Essentials:Utilitites:”) to create copies of the disks.

MCL Documentation

This folder contains the MCL 2.0 documentation, which can be read using Apple DocViewer. A prerelease version of the Apple DocViewer utility and documentation describing it can be found in “Developer Essentials:Utilities:”.

Additional MCL Source Code

This folder contains the high level code for views, dialogs, menus, and the FRED editor. It also includes a file you can load to make edit-definition (meta-.) work for the definitions contained in the code files. This source code is provided as is, and is completely unsupported. If you attempt to program MCL using information gleaned from this source code, we will not be able to provide technical support, nor can we guarantee that your code will work in future versions of MCL.

Unsupported Goodies from Apple

This folder contains additional software from the MCL development team that is not part of the standard MCL product. The contents of this folder are from Apple but are unsupported and may be untested.

User Contributed Code

This large folder contains sample Lisp source code and other things that may be of interest to Common Lisp users. The contents of this folder have been contributed by users and are unsupported and may be untested.

Developer Essentials

Developer Essentials is included on several Apple CD-ROMs for programmers. The folder includes tools and information that may be of interest to Macintosh programmers using any development system. It includes utilities, online versions of Macintosh programming documentation, Macintosh interfaces for several programming languages, and U.S. and international versions of Macintosh system software.

Mail Archives & other docs

This folder includes the info-mcl and comp.lang.lisp archives. It also contains CLIM documentation, miscellaneous information, and OxTeX, a public domain implementation of TeX.

Note: Some text files on the CD-ROM are TeachText documents. However, the TeachText utility (in “Developer Essentials:Utilitites:”) will not open files larger than 32K. If you want to read a text file that is too large to be read by TeachText, you can open it using any application which has an editor or word processor (such as the FRED editor in MCL 2.0). You can either open the file using the application’s “Open…” menu item or, if you’re using System 7, you can open the text file by dragging its icon onto the editing application’s icon (or an alias to the application).

Part 3— Third Party MCL-Based Tools

This section contains information on tools based on MCL 2.0 which are sold by independent developers. The descriptions are provided by those developers.

CLIM, the Common Lisp Interface Manager, is a portable extension to Common Lisp that provides a generic set of user interface capabilities across a wide variety of window systems and GUI environments. CLIM Release 1.1 for MCL 2.0 is a high-performance implementation of the CLIM Version I API standard (defined by a coalition of leading Common Lisp vendors). CLIM's portable features include: abstract coordinate systems and full 2D affine transformations; a generic UI kernel defining basic input, output and windowing operations; input and output character streams; output recording and incremental redisplay, presentation types and input contexts for fully symbolic application user interfaces; a command processor, application frameworks and abstract widget panes; PostScript back-end for screen-hardcopy.

For further information, see the "CLIM (demo)" folder in the "User Contributed Code" folder on the *MCL 2.0 CD*, or contact:

Lucid, Inc.
707 Laurel Street
Menlo Park, CA 94025 U.S.A.

phone: (415) 329-8400
fax: (415) 329-8480
Internet: sales@lucid.com

ExperTelligence's ExperAction! is an interactive interface design tool. Using ExperAction!, a nonprogrammer can create user interfaces, custom screens, and menus, by placing pictures, icons, lists, popup menus, text, buttons, etc. plus user definable objects on the screen quickly and easily. It is as simple as selecting objects from a popup palette and placing them on the screen. ExperAction! separates interface code from functional code, allowing modifications to the interface without affecting (or even knowing) the functional code. ExperAction! is not a code generator; it is a framework that automatically handles the display and behavior of all graphical objects.

ExperTelligence's ExperTools include a class and method browser, a class tree examiner and a documenter that automatically generates completely formatted CLOS class, slot and method documentation in Rich Text Format.

For further information, see the "ExperTelligence, Inc. (demo)" folder in the "User Contributed Code" folder on the *MCL 2.0 CD*, or contact:

ExperTelligence, Inc.
203 Chapala Street
Santa Barbara, CA 93101 U.S.A.

phone: (805) 962-2558
fax: (805) 962-5188
AppleLink: D2042
Internet: D2042@applelink.apple.com

MacYAPS is a power tool for writing expert systems and other software incorporating rule-based programming. It is a flexible and powerful tool for developing knowledge-based systems and other software incorporating data- or state-driven daemonic behavior. The highly flexible syntax of MacYAPS allows a direct mapping from the problem domain into the computer program, enhancing programmer productivity and reducing both knowledge engineering and maintenance costs.

For further information, see the "MacYAPS (demo)" folder in the "Expert Systems" folder in the "User Contributed Code" folder on the *MCL 2.0 CD*, or contact:

College Park Software
461 W Loma Alta Dr
Altadena CA 91001-3841 U.S.A.

phone: (818) 791-9153
Internet: info@grian.cps.altadena.ca.us

Part 4— Upgrading From MACL 1.x

Please read through *New Features of Macintosh Common Lisp 2.0* (in *Getting Started with Macintosh Common Lisp 2.0*).

If you need to convert ObjectLisp code to CLOS, see Appendices D and E in the *Macintosh Common Lisp 2.0 Reference*. In addition, the file "defobfun-to-defmethod.lisp" in the Examples folder contains routines to automate some of the conversion process.

Here is a list of some name changes since MACL 1.*x*. This list is taken from the Examples file "old-dialog-hooks.lisp", which also contains some code to help automate the conversion process:

Old Name	**New Name**
dialog-item-size	view-size
set-dialog-item-size	set-view-size
dialog-item-position	view-position
set-dialog-item-position	set-view-position
dialog-item-nick-name	view-nick-name
add-dialog-items	add-subviews
remove-dialog-items	remove-subviews
find-named-dialog-items	view-named, find-named-sibling
item-named	view-named
dialog-item-dialog	view-container
window-(de)activate-event-handler	view-(de)activate-event-handler
window-click-event-handler	view-click-event-handler
window-mouse-position	view-mouse-position
(set-)window-position	(set-)view-position
(set-)window-size	(set-)view-size
dialog-item-font	view-font
window-font	view-font
:dialog-item-colors initarg	:part-color-list
markp	buffer-mark-p
buffer-char-font	buffer-char-font-spec
buffer-current-font	buffer-current-font-spec
buffer-replace-font	buffer-replace-font-spec
buffer-set-font	buffer-set-font-spec
ed-skip-fwd-wsp&comments	buffer-skip-fwd-wsp&comments
dialog-item-default-size	view-default-size
window-vpos	fred-vpos
window-line-vpos	fred-line-vpos
window-hpos	fred-hpos
window-update	fred-update
window-start-mark	fred-display-start-mark
window-buffer	fred-buffer
catch-abort	use restart-case
catch-error	use handler-case
catch-error-quietly	ignore-errors
color-window-mixin	:color-p initarg
:parent keyword to windows, etc.	:class keyword
add-self-to-dialog	install-view-in-window
remove-self-from-dialog	remove-view-from-window

Fasl Version

You must recompile any files compiled with MACL 1.x.

MCL 2.0 and Lisp Package Compatibility

Steele2 introduced many changes to Common Lisp as it was described in *Steele1*, and a few of the changes introduced incompatibilities between the two.

One Common Lisp change you may encounter early in your conversion process concerns the IN-PACKAGE operation. Previously, this function would create a package with the given name if none existed, then (in any case) put you into it. In the new Common Lisp, IN-PACKAGE is a macro instead of a function, and it signals an error if called with a non-existent package (it will *not* create the package). If you attempt to load a MACL 1.x file that expects the old behavior, you will probably see an error like this one:

```
> Error: There is no package named “FOO” .
> While executing: SET-PACKAGE
```

To create and use a package, you should now begin a file with, for example:

```
(defpackage “FOO” )
(in-package “FOO”)          ; or (in-package :foo)
                            ; but not (in-package 'foo)
```

By default, the :use list is set to (“COMMON-LISP” “CCL”), the value of the variable *MAKE-PACKAGE-USE-DEFAULTS*.

Another change introduced between *Steele1* and *Steele2* concerned the names of the packages in which the language and user code live. In *Steele1*, the code that implemented the language standard was defined in a package called Lisp, and the default package was User. In *Steele2*, the corresponding packages are called Common-Lisp and Common-Lisp-User.

Since the Lisp and User packages were renamed in the new standard, implementations can provide a compatibility mechanism, in which “old” style behavior can be expected in the Lisp and User packages, but “new” style in Common-Lisp and Common-Lisp-User. MCL 2.0 supports such a mechanism. To enable it, either:

```
(require :lisp-package)
```

or, if you want the system simply to take care of it whenever it sees a reference to User or Lisp packages:

```
(setq *autoload-lisp-package* T)
```

After you load the Lisp-package system, IN-PACKAGE (for example) will work in the old way in the Lisp package and in the new way in the Common-Lisp package, but note that no attempt has been made to reproduce all of the behavior of *Steele1* through this mechanism. The source code for this system is in the Library file “lisp-package.lisp”.

Traps

The library of trap definitions has been reorganized: "In ROM" traps (as defined in *Inside Macintosh***)** are dealt with in a different way from "Not in ROM" traps. In addition, new mechanisms for loading them have been introduced to reduce the overhead of using them.

"In ROM" Traps

In past versions of Macintosh Common Lisp, calls to Macintosh traps could be made only after requiring that all trap interfaces be loaded, even if you only intend to use a subset of these interfaces. Thus, for example, "ccl;library:quickdraw.lisp" used to begin:

```
(eval-when (eval compile)
        (require 'records)
        (require 'traps))
```

In MCL 2.0, there are two new reader macros (#_ and #$) that cause specific "In ROM" traps to be loaded automatically when they are called. Thus, "quickdraw.lisp" no longer begins with these REQUIREs, and all the trap calls are now preceded by "#" (eg, #_NewRgn instead of _NewRgn). There are a few other changes in trap syntax as well. They are documented in Chapter 17 of the Reference Manual.

"Not In ROM" Traps

Most "not-in-ROM" traps have been moved into the file "CCL:Library;interfaces.lisp". They cannot be autoloaded using the "#_" syntax. Michael Engber of Northwestern University's Institute for the Learning Sciences has provided a uniform syntax for these traps as well. The NotInROM folder in the Examples folder contains the mechanism, along with documentation (in "NotInROM-docs.txt"), from which this brief summary was drawn:

> NotInROM provides a way to define trap calls which MCL does not provide. These are generally traps that are listed as "Not in ROM" in *Inside Macintosh***.** To call a "Not in ROM" trap you should use the following syntax:
>
> ```
> (#~SomeTrap argsI)
>   or
> (require-trap-NotInROM #~SomeTrap argsI)
> ```

The “In ROM” traps have been defined as documented in *Inside Macintosh*. Some of the “Not In ROM” definitions have been given a syntax that is more natural for Lisp programming than they had in previous versions. These definitions were originally translated from Pascal trap definitions, and in some cases used Pascal conventions that make little sense in a Lisp environment. Pascal functions cannot easily return multiple values, and they often use VAR parameters (where pointers are passed as arguments simply so that the called function can store values there for later use) in places where a Lisp programmer would simply have the function return multiple values. Many of the trap calls faithfully copied Pascal’s use of VAR parameters. The old definition of _screenres, for example, took two arguments:

```
(deftrap _screenres   (  (scrnhres (:pointer :signed-integer))
                         (scrnvres (:pointer :signed-integer)))
 nil
 (:no-trap (progn
                  (%put-word scrnvres (%get-word (%int-to-ptr #x102)))
                  (%put-word scrnhres (%get-word (%int-to-ptr #x104))))))
```

Note that screenres doesn’t need the values of its arguments, it simply needs a place to put a couple of values. In MCL 2.0, screenres is one of the “Not In ROM” definitions that have been moved to “Library:Interfaces.lisp”, where it is defined as a function. It simply returns the two values:

```
(defun screenres ()
        (values (%get-word (%int-to-ptr #x104))
                 (%get-word (%int-to-ptr #x102))))
```

Trap Names

The trap files generally contain both the assembly language and the Pascal forms of trap names. So, for example, GetEOF and PBGetEOF are both defined and are defined identically. Many of the assembly language names for “In ROM” traps are identical to the Pascal names for “Not-in-ROM” traps. MCL’s “#_” reader macro will find the assembly language “In ROM” traps in these cases. For example, #_GetVol is the same as #_PBGetVol.

A few other details

Some MCL environment function names have changed (eg, BUFFER-INSERT-STRING is now BUFFER-INSERT-SUBSTRING). Please check the documentation if you encounter “undefined function” errors.

The class COLOR-WINDOW-MIXIN no longer exists; use the :COLOR-P initarg when creating a window instead.

WINDOWS, MAP-WINDOWS, and FRONT-WINDOW now take keyword arguments (:CLASS, :INCLUDE-INVISIBLES, and :INCLUDE-WINDOIDS) instead of &optional arguments.

Part 5— Upgrading from MCL 2.0b1

This section points out new features and changes to existing features in a fairly general way. Please see the *Reference Manual* and *Steele2* for further information. Also, please skim the section "Upgrading from MACL 1.x to MCL 2.0" earlier in this document, as no attempt has been made to duplicate below any material that appeared there.

What's New

Here is a partial list of new features to be aware of in MCL 2.0:

- Ephemeral Garbage Collector (EGC).

- Complete *Steele2* compatibility including integrated pretty printer (the "XP" folder is no longer required)

- CLOS includes the following introspection capabilities
  - class-direct-subclasses
  - class-direct-superclasses
  - class-precedence-list
  - class-prototype
  - class-direct-instance-slots
  - class-direct-class-slots
  - class-instance-slots
  - class-class-slots
  - specializer-direct-methods
  - specializer-direct-generic-functions
  - generic-function-methods
  - method-function
  - method-generic-function
  - method-name
  - method-qualifiers
  - method-specializers
  - slot-definition-name
  - copy-instance
  - method-exists-p

- Short-float is now an "immediate" (non-consing) datatype. See Appendix A of the Reference Manual for details.

- Read-only files in FRED: If you open a file that is locked or marked as read only by Projector, FRED will now open it in a read only buffer. The title is marked with a special glyph (an "R" in a circle) and any attempt to modify the buffer will be rebuffed.

- System 7.0 support:
  - Balloon Help
  - Aliases
  - AppleEvents

- ToolServer interface

- Improved and more complete ToolBox interfaces

- New hash table implementation with improved locality, and better garbage collection interaction.

- New Loop conforms to the *Steele2* specification. (The “old” loop is still distributed, however, as “MIT-loop.lisp”. If your MCL 2.0b1 code does not work with Loop, you can simply rename “MIT-Loop.lisp” to “Loop.lisp” and it will be autoloaded in place of the new loop.)

- CD-ROM including
  User contributed code
  Info-mcl mailing list archive
  MCL Reference online in DocViewer format

Here is a little detail on some of the changes you may notice:

Fasl Version

You must recompile any files compiled with most previous versions of MCL 2.0, including MCL 2.0b1. If you are in doubt about whether you need to recompile, simply try loading a compiled file: MCL 2.0 will signal a “Wrong fasl version” error if it cannot load the file.

Apropos-Case-Sensitive-P

Apropos is now, by default, case-insensitive. Thus:

```
? (defvar xyz 123)
XYZ
? (apropos “xyz”)
XYZ,  Value: 123
```

This behavior can be changed by setting the variable CCL::*APROPOS-CASE-SENSITIVE-P* to a true (non-nil) value.

Stepping and Tracing

In MCL 2.0b1, if you stepped through a function that had been defined with:

compile-definitions t
save-definitions t

the uncompiled definition would be installed in the function (or method): the stepped function would “lose” its compiled definition and be interpreted forever after. In MCL 2.0 this is no longer true; the stepper uses the saved definition for stepping, but the function is still compiled when called normally.

Similarly, stepping a generic function caused all its methods to be uncompiled in MCL 2.0b1. In MCL 2.0 this is no longer true. Methods must have been defined with *compile-definitions* nil, or must be individually “trace stepped” to be used in the stepper. For example:

(trace ((:method foo (specializer)) :step t))

Starting up MCL 2.0 by double-clicking on a document icon

In MCL 2.0, double-clicking on the icon of a fasl (compiled) file causes it to be loaded, after starting up MCL 2.0 if necessary. Double-clicking on a lisp file causes it to be read into a FRED window.

A few other details

Accented characters are considered alphabetic by CHAR-UPCASE, CHAR-DOWNCASE, ALPHA-CHAR-P and the FRED commands for upcase, downcase, and capitalize.

SAVE-APPLICATION no longer takes a :COMPRESS keyword argument.

Common Lisp logical pathnames are now fully supported, and logical directories are no longer defined.

FRED handles mode lines and in-package references in a different way from that of 2.0b1. It is documented in Chapter 2 of the Reference Manual.

The trap #_HandToHand has been changed to agree with its documentation in *Inside Macintosh*. In MCL2.0b1 , it returned a handle; it now takes a single VAR parameter which it uses for both the handle coming in and the handle going out, and returns an error code as the function result.