Digitool

*For Macintosh Common Lisp versions 3.1 & 4.0*

# Macintosh Common Lisp Release Notes

Digitool

Developer Technical Publications

**Digitool**, Inc.



Printed in the United States of America.

MCL is a trademark of Digitool, Inc.

One Main Street,
Cambridge, MA 02142
617-441-5000



Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014-6299
408-996-1010



Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, Digitool will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to Digitool.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Digitool has reviewed this manual, **DIGITOOL MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL DIGITOOL BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL,** even if advised of the possibility of such damages.

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED.** No Digitool dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

# Contents

# MCL 4.0/3.1 Release Notes

These notes describe changes made in the most recent versions of MCL.

If you will be programming on a PowerPC Macintosh, you will be using MCL 4.0. The changes described are relative to the previous PowerPC release, MCL 3.9.

If you will be programming on a 68K Macintosh, you will be using MCL 3.1. The changes described are relative to the previous 68K release, MCL 3.0.

If you are upgrading from the 68K version of MCL to the PowerPC version, you should also read the MCL 3.9 Release Notes, which are included in the MCL 3.9 documentation folder. These release notes describe MCL's transition from the 68K to the PowerPC.

For the most part, MCL 4.0 and MCL 3.1 provide the same programmer's interface. Differences are noted in a final section of these release notes.

## Installation Instructions

Complete configuration requirements and installation instructions for MCL 4.0 and 3.1 are given in *Getting Started with MCL*, available in hardcopy and Acrobat format with this release.

Note that the PPCExceptionEnabler system extension is no longer needed by MCL 4.0.

## MCL Demo

The MCL CD now includes a demonstration version of MCL. This version includes all the features of the full version of MCL, except that it can only be used for fifteen minutes at a time. After fifteen minutes, it will automatically quit to the Finder.

Trial users can write to Digitool for a password which will allow them to use the demonstration for longer than fifteen minutes at a time. The password lasts for thirty days. To receive a password, send e-mail to demo-mcl@digitool.com.

The demonstration version has the MCL libraries built-in. They do not need to be included separately. However, there must be enough free memory to load the libraries. (If there is not, the Macintosh will give an erroneous "library not found" message.)

Please feel free to share this demonstration version of MCL with your friends and colleagues. Trial users can access the MCL Getting Started guide on the world wide web at

http://www.digitool.com/MCL-getting-started.html

# Changes Common to MCL 3.1 and 4.0

This section describes changes which were made in both MCL 4.0 and 3.1.

## Compiled Files

The fasl version has changed in both MCL 4.0 and MCL 3.1. Because of this, all files compiled in an earlier version of MCL will need to be recompiled.

## Save Application

The default value of the `:init-file` argument to `save-application` is no longer `nil`. Instead, it is the result of calling `application-init-file` on `*application*`. If you do not want any init file loaded, you should ensure that this call returns `nil` for your program's value of `*application*`.

## Logical Directories

Old style logical directories (as created by the function `def-logical-directory`) are no longer built-in to MCL. To ease the transition for people who still use them, the file "Library;logical-dir-compatibility.lisp" can be used to restore the functionality.

## Table Drawing

`set-table-sequence` now invalidates the entire table only if the new sequence is not `eq` to the old one and at least one of the elements is not `eq`. This change was made to stop the flashing in some dialogs. If you have code that modifies a table's sequence in place and then calls `set-table-sequence` to display the modified sequence, the display may not update as you wish. You'll have to change the code to explicitly redraw as necessary.

If you wish to restore the old behavior, you can do so with the following method:

```
(defmethod set-table-sequence :after ((item t)
                                      (new-sequence t))
  (invalidate-view item))
```

## Handles

`with-dereferenced-handles` is no longer a synonym for `with-pointers`. `with-dereferenced-handles` now assumes that the handles it is passed are really handles. This should require no changes to code that was written according to the documentation of `with-dereferenced-handles`. It may, however, break code that inadvertently calls `with-dereferenced-handles` with non-handles.

## Floating Point Access

`:double-float` and `:single-float` now work as record field types. Accessing these record fields returns a freshly allocated floating point number.

`%get-double-float`, `%get-single-float`, `%hget-double-float`, `%hget-single-float`, `%put-double-float`, `%put-single-float`, `%hput-double-float`, `%hput-single-float` are new accessors that work just as `%get-long` and friends, but access floats.

These getters take an optional floating point number parameter that will be destructively modified and returned, so that it is possible to access a float without consing. If you use this feature, be careful not to accidently overwrite a floating point number that you are using elsewhere. Take particular care not to override the system-wide unique 0.0 instance. It is safest to create a fresh target by explicitly allocating a new float, for example by the call `(ccl::%copy-float 0.0)`.

## Script Manager Extensions

The following extensions to the script manager are present in both MCL 4.0 and MCL 3.1.

### `set-extended-string-script` [*Function*]

**Syntax** `set-extended-string-script` *script*

Sets the script to use for printing extended-strings. If the script is not set explicitly, the default is the system script if it is a 2 byte script; otherwise the default is an installed 2 byte script. If there are no installed 2 byte scripts, the default is `nil`.

**Arguments** *script* A script, as described by Inside Macintosh.

### `set-extended-string-font` [*Function*]

**Syntax** `set-extended-string-font` *font-spec*

Sets the font to use for printing extended strings. If not set explicitly, the default is the `#$smScriptAppFond` for the extended-string script.

**Arguments** *font-spec* A font spec, as described by the MCL Reference Manual.

---

### `convert-kanji-fred` [*Function*]

**Syntax**

```
convert-kanji-fred oldpath &optional
                                newpath (if-exists :supersede)
```

This function is used to convert files produced by KanjiFred to a format that can be used by MCL 4.0 and 3.1.

**Arguments**

*oldpath* The pathname of the file to convert.

*newpath* The pathname at which to store the converted file. The default is *oldpath*.

*if-exists* A keyword describing what action to take if *newpath* already exists. The allowed keywords are the same as for `copy-file`.

---

### `*input-file-script*` [*Variable*]

This variable can be set to a 2 byte script to be used when reading a text file that was not created with Fred and is known to contain text in a single 2 byte script.

---

## Miscellaneous New Functions

The following new functions are present in both MCL 4.0 and MCL 3.1.

---

### `fred-autoscroll-h-p` [*Generic Function*]

### `fred-autoscroll-v-p` [*Generic Function*]

**Syntax**

```
fred-autoscroll-h-p (view fred-mixin)
fred-autoscroll-v-p (view fred-mixin)
```

These generic functions are called indirectly by `(view-click-event-handler fred-mixin)`. They control whether the view will be automatically scrolled when the mouse cursor goes outside of it. The default methods return true. You can specialize this generic function on subclasses of `fred-mixin` in order to change this behavior.

**Arguments** *view* A `fred-mixin`.

---

## `load-patches` [*Function*]

**Syntax** `load-patches &optional` *source-dir all*

Loads some or all of the compiled files in the patch file directory, and optionally sets a patch version number which determines the version specified in the `vers 1` resource created when `save-application` is called. The patches directory is a folder whose name is of the form "Patches *x*.*y*", where *x* and *y* are the major and minor version numbers of MCL (for example, "Patches 3.1b1" or "Patches 4.0").

If *all* is `nil`, only new patches are loaded. A patch is considered to be new if its name (excluding file extension) ends in "p*n*", where *n* is a number greater than the current patch version. The current patch version is determined from the `vers 1` resource. The patch version number will be set to the highest value of *n* encountered, and is returned by `load-patches` if set. This value is then used by `save-application`.

If *all* is true, all patches are loaded and the patch version is not set.

**Arguments**

*source-dir* The directory containing the patch file directory. The default value for this argument is the value of the form `(full-pathname "ccl:" :no-error nil)`.

*all* If true, load all compiled files in alphabetical order, and don't set the patch version number. If `nil` load only compiled files with names as specified above, and set the patch version number. The default is `nil`.

---

## `load-all-patches optional source-dir` [*Function*]

**Syntax** `load-all-patches &optional` *source-dir*

Loads all compiled files from a patches directory by executing `(load-patches` *source-dir* `t)` and resets the current patch version to `nil`. Returns `nil`.

**Arguments**

*source-dir* The directory containing the patch file directory. The default value for this argument is the value of the form `(full-pathname "ccl:" :no-error nil)`.

## `application-init-file` [*Generic Function*]

**Syntax**

```
application-init-file (application application)
application-init-file (application
                       ccl::lisp-development-system)
```

This generic function is used to specify the init-file of an application. The built-in method specialized on the `application` class returns `nil` (meaning there is no init file). The built-in method specialized on the `ccl::lisp-development-system` class returns `"init"`.

This generic function is used to generate a default value for the `:init-file` argument of `save-application`.

**Arguments**

*application* An application class.

## Exported Symbols

The following additional symbols are exported from the CCL package.

- `add-to-shared-library-search-path` (4.0 only)
- `application-init-file`
- `convert-kanji-fred`
- `fred-autoscroll-h-p`
- `fred-autoscroll-v-p`
- `%get-double-float`
- `%get-single-float`
- `%hget-double-float`
- `%hget-single-float`
- `%hput-double-float`
- `%hput-single-float`
- `*input-file-script*`
- `load-all-patches`
- `load-patches`
- `%put-double-float`
- `%put-single-float`
- `remove-from-shared-library-search-path` (4.0 only)
- `set-extended-string-font`
- `set-extended-string-script`

## Menu and Tool Changes

The following changes to menus and tools were made in both MCL 4.0 and MCL 3.1.

### Search Files Tool

The user interface for the Search Files tool has changed. If you prefer the old user interface, set `ccl::*use-old-search-file-dialog*` to `t`.

### Text Editing Menus

A set of font menus has been added to the Edit menu. The source code for these menus is in the file "font-menus.lisp". This file has been moved from the "Examples" folder to the "Library" folder.

There is a new Word Wrap command on the Edit menu. This command allows you to turn word wrap on and off in the current editor.

### Extensions Menu

An "Extensions" submenu has been added to the Tools menu. The menu allows you to load individual or aggregate facilities, and to save a new image quickly.

### Listeners

Listeners created for error break loops now indicate the name of the process they represent in their titles.

### Stack Backtrace Tool

The stack backtrace now hides the internal frames. To show them, either select the Show all frames command or the Default show all frames command on the Commands pop-up in the Stack Backtrace tool, or set `inspector::*backtrace-hide-internal-functions-p*` to `nil`.

If you wish the backtrace to hide additional functions, add them or their names to `inspector::*backtrace-internal-functions*`.

### Apropos Tool

The Apropos Tool accessed through the Tools menu no longer has Specializers or Qualifiers pop-ups. It has new check box for External Symbols Only.

### Trace Tool

The Trace tool has a new pop-up which allows the traced function to be stepped.

### Execute Buffer Command

The Execute Buffer command on the Lisp menu has been renamed to Execute All.

### Balloon Help

Due to an apparent bug in the `_HMShowBalloon` system call, balloon help will not be shown for a menu-item if the menu-item is above the active window. MCL can work around this bug by temporarily setting the low-memory global `#$windowlist` to null while showing the balloon help. If you want MCL to do this, set the variable `*fix-menu-balloon-help-bug*` to a true value. The default value of this variable is `nil`.

## Changes Specific to MCL 4.0

The following changes made to MCL 4.0 were not made to MCL 3.1.

### Memory Requirements and Startup

When run without virtual memory, MCL 4.0 requires half as much memory for its heap as 3.9 did. It also starts up faster with virtual memory enabled, and the application disk file is smaller.

(These features first appeared in the 3.9.1 kernel, which was made available to beta testers.)

## Garbage Collection

The ephemeral garbage collector now works in MCL 4.0, and the functions that control it are the same as in earlier versions of MCL. See the reference manual descriptions of `egc`, `egc-enabled-p`, `egc-active-p`, `egc-mmu-support-available-p`, `egc-configuration`, and `configure-egc`.

## Stacks and Processes

In MCL 4.0, two out of the three stacks for a stack group are now segmented. If a stack overflow occurs in one of those two stacks, you can continue from the resulting error just as you can in MCL 3.1. The `:stack-size` keyword argument to `make-process` and `process-run-function` and the optional *stack-size* argument to `make-stack-group` control the total amount of stack space that will be allowed to accumulate before an error is signalled. A stack group's control stack, the one allocated by the Macintosh thread manager, is not segmented. An overflow of the control stack always signals an error. The initial size of the control stack is 1/3 of the `:stack-size` argument to `make-process`, `process-run-function`, or `make-stack-group` except that it will never be less than `ccl::*min-sp-stack-size*`. The initial value of `ccl::*min-sp-stack-size*` is 16K.

## Save Application

`save-application` now copies all the `cfrg` resources from the application file. This allows you to bundle the CCL application and the pmcl-kernel, pmcl-compiler, and pmcl-library shared library files into a single file. It also allows you to add your own shared libraries to the application file. The file "Examples;cfm-mover.lisp" includes functions that bundle all the MCL libraries into a single file (see the commented out example form at the bottom of the file).

The `cfrg` resources are generated automatically `save-application`. If you specify a resource file using the Save Application tool, or if you specify a `:resources` argument to `save-application`, you should not include a `cfrg` resource.

The `:excise-compiler` option to `save-application` now works in MCL 4.0.

## Interpreted Foreign Access

MCL 4.0 code that calls external functions defined with `deftrap`, `define-entry-point`, or `defpascal` needs to be compiled if it is to run in an application with the compiler excised. Attempting to interpret such functions will invoke the compiler, and error if the compiler is not present.

## Optimizations

MCL 4.0 contains a number of significant optimizations that were not present in MCL 3.9.

- Logical operations (and arithmetic operations where the result is known to be a fixnum) on more than 2 known fixnums are now performed inline. They used to be inlined only for exactly 2 operands.
- The compiler now recognizes that the result of `+`, `-`, `/`, and `*` of two double-floats is also a double float. Operations with more than 2 double-float arguments are performed inline.
- Some arithmetic operations are executed by subprimitives that check for fixnum operands and if so do the operation without calling the general function.
- `format` of floats is faster and conses less.
- `truncate`, `floor`, `ceiling`, and `round` no longer compute their second return value unless it is required by the caller.
- `truncate`, `floor`, `ceiling`, and `round` of bignums are faster and cons less. `rem` and `mod` of bignums do not cons the first value of `truncate`. Some other bignum operations cons less.
- `equal` is faster.
- Some cases of `typep`, `require-type`, and `find-class` are optimized.
- `sort` no longer calls the key function more often than necessary.
- Some additional optimizations in hash table access are present.
- Echo of typing in the editor is faster.
- Some string manipulation functions are faster.
- Equal hash tables are faster for list or string keys that contain the same elements in differing order.
- There was an error in printing circular structure that caused it to be extremely slow.
- `vector-push-extend` grows the vector by a fraction of its size rather than a constant value.

- Generic functions of 1 argument or 2 arguments specialized on the first are faster. There are some method dispatch improvements. `class-of` is faster.
- `make-point` is faster.
- There are now compiler optimizers for `logorc2`, `lognand`, `lognor`, `logandc2`, and `lognot`.
- Some cases of `assoc` and `member` are faster.

## Changes Specific to MCL 3.1

A large number of bugs have been fixed between MCL 3.0 and MCL 3.1. Most of these bug fixes were previously released as patches to MCL 3.0. For lists of specific patches, see the patch release notes.

## Differences Between MCL 4.0 and MCL 3.1

- The full termination facility implemented by `terminate-when-unreachable` and related functions is available in MCL 4.0 but not 3.1.
- MCL 3.1 has short floats, 4.0 does not.
- MCL 4.0 supports shared libraries and accesses system functionality through shared library entry points. MCL 3.1 Does not support shared libraries, and it accesses system functionality through traps. (While the syntax of shared library entry point calls and trap calls are the same, the compilation environments are slightly different. See the MCL 3.9 release notes for details.)
- There are minor differences in the type systems of MCL 4.0 and 3.1.
- In MCL 4.0, arithmetic conditions include the operation and arguments. They are not included in MCL 3.1.
- MCL 4.0 provides user control over which floating point exceptions to signal.

- Stack allocation is different in MCL 4.0 and 3.1.
  In MCL 4.0 a process has 3 stacks. The value stack and temporary stack are allocated in multifinder temporary memory if available, otherwise in the Mac heap; these stacks can be grown dynamically. The control stack is allocated by the Macintosh process manager on the Mac heap; this stack cannot be grown dynamically.
  In MCL 3.1 a process has 2 stacks allocated in the Mac heap. Both of these stacks can grow.
- MCL 4.0 inlines more arithmetic operations when argument types are declared.
- Fixnum size is 30 bits in MCL 4.0, and 29 bits in MCL 3.1.
- Array size limits are different in MCL 4.0 and 3.1. In 4.0 an array can have 16 million elements, while in 3.1 it can have one million elements.

# Index