DRAFT 12/21/22

MPW PQR4 Shell
Release Notes

This "provisional" version of the MPW Shell contains the following new features:

- Several modifications to the Editor affecting the actions of keyboard controls for moving the cursor and deleting text, and permitting customization of these controls.
- An enhancement to Projector, allowing the optional storing of the complete revision history in the *ckid* resource of a file.
- A pair of new Projector commands, `ObsoleteProjectorFile`, and `UnObsoleteProjectorFile`, which respectively remove a file from active participation in a project and which reverse the removal.
- A command `RenameProjectorFile`, which enables the user to effect name changes.
- An extension to the syntax of the Shell scripting language, simplifying the processing of parameter strings containing arbitrary numbers of single and double quotes.
- A new Shell variable, `TraceFailures`, aids the debugging of scripts.
- Under System 7, the Shell is "stationery aware."

Editor

The keyboard commands for moving the insertion point, deleting text, and extending selections have been modified for consistency and conformity with Human Interface Guidelines. There are additional commands which extend the Guidelines especially for MPW users. User testing of these extensions have not yet been completed, so the functionality **may** change in the next release. Therefore, it is premature to consider these keyboard commands as guidelines for new applications. We are interested in your feedback on this topic. Please send comments to MPW.BUGS via AppleLink. Two new commands, **`SetKey`**, and **`UnSetKey`**, make to possible to redefine these and to define entirely new keyboard commands.

SetKey, UnSetKey

The purpose of these commands is twofold: allowing the user to change the keyboard layout of the editor primitives such as MoveCharLeft, DeleteEndOfLine; and allowing the user to configure the editor so that it behaves similarly to a favorite line editor such as Emacs or vi.

Syntax

`SetKey` <*empty*> | `[modifierkey-]…key[-[modifierkey]…key] [command]`
`UnSetKey [modifierkey-]…key[-[modifierkey]…key]` [*any text here is ignored*]

The modifier keys are:
`command option shift control`

The keys are:
`clear del delete downarrow end enter escape help home leftarrow pagedown pageup return rightarrow space tab uparrow ' , - . / 0 1 2 3 4 5 6 7 8 9 ; = [ \ ] ` a b c d e f g h i j k l m n o p q r s t u v w x y z f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15`
(On numeric keypad) `kp*` `kp/` `kp+` `kp-` `kp.` `kp0` `kp1` `kp2` `kp3` `kp4` `kp5` `kp6` `kp7` `kp8` `kp9` `kp=`

Examples:
SetKey f11 'MoveWindow {LeftPos} "{Active}"; SizeWindow {LeftPos} "{Active}"
SetKey f12 'MoveWindow {RightPos} "{Active}"; SizeWindow {RightPos} "{Active}"
SetKey control-x-y 'Execute `Request "Enter a Command"`'

The first two of the above bind the function keys `f11` and f12 to scripts that move a window. The third provides for execution of a command line without altering the active window. Note that `control-x-y` means the sequence "control-x" followed by "y."

If `SetKey` is executed without any parameters, it prints a list of commands that would restore the keyboard to its state at the time of execution. If it is executed with a key description but without a command, it prints the commands currently assigned to that key. A key can also be specified by the raw keycode in hex (e.g. `$32`). Modifier keys can also be specified in hex using the keycode format documented in Inside Macintosh Vol V, page 195 (the capslock bit is ignored).

Default Key Assignments

The entire set of default key assignments can be printed by executing the command SetKey.

The following table shows the majority of them.

	delete	**del** *	←	→	↑	↓
	deletes character to left	deletes character to right	moves character to left	moves character to right	moves one line up	moves one line down
shift	deletes character to left	deletes character to right	extends selection character to left	extends selection character to right	extends selection one line up	extends selection one line down
option	deletes word to left	deletes word to right	moves word to left	moves word to right	moves one line up	moves one line down
shift option	beep	beep	extends selection word to left	extends selection word to right	extends selection one line up	extends selection one line down
cmd	deletes to end of file	deletes to end of file	moves to beginning of line	moves to end of line	moves one page up	moves one page down
cmd shift	beep	beep	extends selection to beginning of line	extends selection to end of line	extends selection one page up	extends selection one page down
cmd option	deletes to end of file	deletes to end of file	moves to beginning of line	moves to end of line	moves to beginning of file	moves to end of file
cmd shift option	beep	beep	extends selection to beginning of line	extends selection to end of line	extends selection to beginning of file	extends selection to end of file

* **del** is available on extended keyboard only

Editor Primitives

It should be emphasized that `SetKey` can assign to a key or a key sequence two kinds of names: scriptable names such as built-in commands, script names, or tool names, and editor primitives, which are currently non-scriptable. It is, however, the long-range intent to make the editor primitives scriptable. The names of the editor primitives—the correspondence to the actions given in the above matrix as default key assignments is obvious—are:

```
      DeleteCharLeft DeleteCharRight DeleteEndOfFile DeleteEndOfLine
      DeleteStartOfFile DeleteStartOfLine DeleteWordLeft DeleteWordRight
      MoveCharLeft MoveCharRight MoveEndOfFile MoveEndOfLine      MoveLineDown
MoveLineUp MovePageDown MovePageUp MoveStartOfFile    MoveStartOfLine
MoveWordLeft MoveWordRight SelectCharLeft       SelectCharRight
SelectEndOfFile SelectEndOfLine SelectLineDown  SelectLineUP SelectPageDown
SelectPageUp SelectStartOfFile      SelectStartOfLine SelectWordLeft
SelectWordRight
```

Projector

History with CheckOut

Projector now has a facility for storing in the *ckid* resource of a file all of the information that is normally displayed by calling `ProjectInfo`: author, checkin date, task, and comment. The facility is invoked either by using the syntax

```
checkout -history file[,n]
```

or by marking the check box labelled “Keep history” in the CheckOut window. The purpose of the facility is to make it possible to obtain the history of a file without mounting its project.

The history that is stored with a file is for those revisions which are the file’s ancestors. Thus, if the checked out revision is on a branch, the history will be for the revisions on a direct path from that revision back to the root (the initial revision).

There are two ways to display this information:

The first is to use

```
projectinfo -comments name
```


This will normally display the checkout data for the revision, followed by a full checkin history for that revision and its ancestors. This display format is independent of whether or not the project is mounted at the time of the command unless *name* is a leaf name (e.g. `MyFile`) in contrast to a full or partial path name (e.g. `HD:MPW:MyFile` or even `:MyFile` if `HD:MPW` is the current directory). If the project is mounted and the name is a leaf name, then only the checkin history is given, the data coming from the project and not from its counterpart in the file.

The second is to click on the "?" in the CheckIn window and to select the file. This will cause the entire history to appear in the Comment box regardless of whether or not the project is mounted.

Obsoleting a File

A new command, `ObseleteProjectorFile`, causes a file within a project to become inactive. This means:

- The file cannot be checked out for modification. No further revisions of it can be created.
- A "checkout all" or "checkout newer" will not check out this file.
- Existing revisions of the file can be checked out read-only. A file which has been named using `NameRevisions` and subsequently made obsolete will still be included in a selection by name provided that the checkout is read-only.

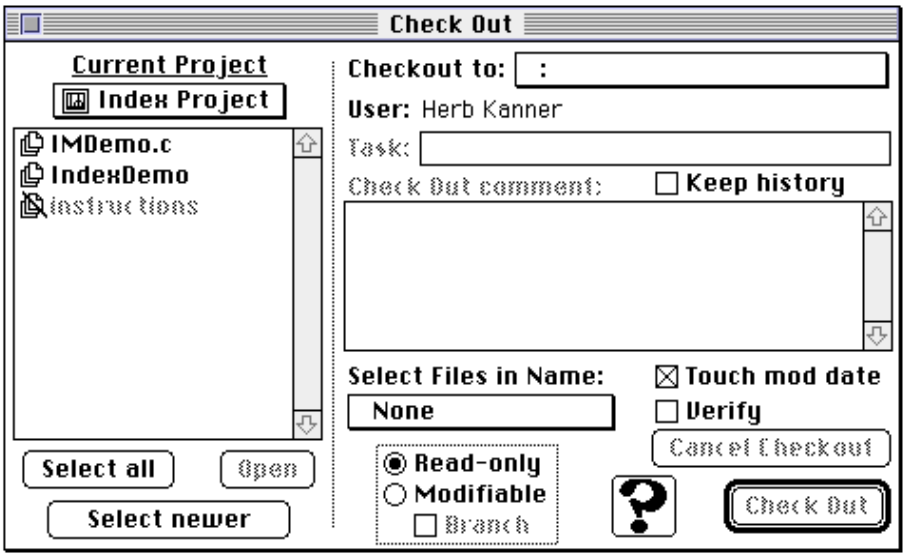
The syntax is:

```
ObsoleteProjectorFile [-p] [-u User] [-project Project] file…
-p                 # write progress information to standard output
-u User            # name of current user
-project Project   # name of project that contains the files
```

The following statements are true about an obsoleted file, e.g. file.c:

- `Checkout file.c` will fail.
- `Checkout -a` will not check out file.c.
- `Checkout -obsolete file.c` will check out read-only the last revision of file.c.
- `Checkout -obsolete file.c,3` will work if revision 3 exists.
- Checkout Beta1 will check out file.c if it was part of the set Beta1.
- `ProjectInfo` will include the info for obsoleted files with the flag "`(Obsolete)`" on the lines bearing such info.

Obsoleted files are indicated in the Checkout window by an icon with a line through it and by the file name being dimmed, as is shown in the following picture.



In the Checkout window:

- Selecting **all** or **newer** will never select obsoleted files.
- Obsoleted files can be selected by option-clicking them.
- Obsoleted files can never be checked out modifiable.
- Selecting a Name will select obsoleted files if they are part of the named set.

△ **Important** All people sharing the use of a Projector database should be using MPW Shells which are the same with respect to this command. If some people are using an old Shell, which does not know about obsoleted files, they will be able to modify and check in a new revision of such a file. To the user of a new Shell, the file is still obsolete, but a new revision has mysteriously appeared. △

The process which obsoleted a file can be reversed by using the command `UnObsoleteProjectorFile`.

The syntax is:
```
UnObsoleteProjectorFile [-p] [-u User] [-project Project] file…
```

Renaming a File

The following command will rename a file:
```
RenameProjectorFile [-p] [-u User] [-project Project] oldName newName
```

The name is changed in the database. The old name is, in fact, forgotten and could be used as the file name when checking in an entirely new file. When listing NameRevisions, the new name will appear in the output. The only possible source of confusion is that the NameRevisions commands in the log will still have the old name in them.

Scripting Language

A new syntax for denoting the value of a variable and the output from a command has been defined. For variables, the syntax is `{{SomeVariable}}`; for command output, it is ``` ``SomeCommand`` ```. The distinction between this and the old syntax, `{SomeVariable}` and `` `SomeCommand` `` is that the new syntax causes all single and double quotes in the value or output, respectively, to be escaped, so that the quotes are taken as literal characters.

For example, consider `Set aVariable 'ab"c'`. MPW defies any attempt to display this variable via `Echo`. `Echo {aVariable}` or even `Echo "{aVariable}"` produces the diagnostic information that "s must occur in pairs. However, `Echo {{aVariable}}` yields the original string.

As a more difficult example, let the file `someFile` contain the text:
```
"Resource" identifiers can be placed in the code with single (').
```

`Set aSelection "``Catenate someFile``"` followed by `Echo {{aSelection}}` will display the contents of the file. Use of the old syntax in any of the above will either produce a diagnostic about the unbalanced single quote, or will lose the double quotes around the word `Resource`.

This syntax makes it possible to do previously impossible things in scripts. It should be said, however, that it will probably be useful only to those who are very experienced in the use of the scripting language.

Shell Variable for Script Failures

A new shell variable, `TraceFailures`, aids in analysing script failures. If this variable is set to `True`, then when a script terminates abnormally, a message identifying the location of the failure, similar in form to compiler diagnostic messages, is sent to diagnostic output.

Stationery

The Shell is "stationery aware." This means that the process of opening a stationery pad appears to the user to be no different from the opening of a "new" document, except that the initial contents of the window will be a copy of the stationery pad.