Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

MPW QR4 Appendix G

Release Notes

PROJECTOR

An Informal Tutorial

Introduction

Projector is an integrated set of tools and scripts whose primary purpose is the control of source code. The system has two basic functions. The first is to make it practical to have several people working simultaneously on a project. By allowing only one person at a time to modify any given file, it prevents a programmer from inadvertently destroying changes made by another. The second function is to preserve, in an orderly manner, revisions of a file and commentary on the revisions. This enables programmers to find out the author and revision date, to read the revision commentary, and if desired, to retrieve the revision itself.

Projector represents a considerable advance over older systems, such as SCCS and its descendants, known to users of UNIX®. It uses its own window interface for three of its commands (NewProject, CheckIn, and CheckOut). These windows, unlike Commando windows, can stay open indefinitely. The Commando interface is also available for all commands, although its use is not recommended for the three commands just mentioned. Projector also differs from SCCS in that its use is not restricted to text files. However, the data compression achieved by storing only one complete copy of a file and storing revisions as files of differences is only available for text files. Projector also has a degree of flexibility which permits different users of the same set of files to view them differently. This is accomplished by giving each user independent control of the mapping between the local directory hierarchy into which he/she keeps the files and the hierarchy used for their storage in the Projector database. Finally, Projector has a facility for associating a specific set of file revisions with a name, this name being usable as a designator for a particular version, or release, of a product. Thus, the name alone can be used to trigger the selection of just those sources that are required to build the desired instance of the product.

This tutorial begins with a section that discusses the basic concepts and terminology. Following this are sections that demonstrate the use of Projector by creating a database skeleton from scratch, putting files into it, and performing various revision activities. These sections are illustrated with screen shots taken during the actual operations.

Concepts and Terms

The top level, fundamental construct in a Projector database is called a *project*. Projects are analogous to directories in an HFS (hierarchical file system). A project may contain files and may contain other projects; just as a directory may have sub-directories, a project may have subprojects. The difference is that a file name in a project represents all revisions of the file—this is known as a *revision tree*—and is also a pointer to *file information* and *revision information*. File information is descriptive text that applies to all revisions, while revision information is descriptive text that relates to a single revision. Fig. 1 illustrates the project hierarchy that will be used throughout

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

the next chapter.

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

The symbol "∫" is used as a separator in naming projects much as the symbol ":" is used in hierarchical file names. Thus, Base∫ is a project, Base∫Sources∫ is a subproject of Base∫, and Base∫Sources∫C∫ is a subproject of Base∫Sources∫. As is the case with directories, the terminal separator may be omitted, e.g. Base∫Sources. Although there is a parallel concept to that of *current directory*, namely *current project*, there is no provision for a partial project name relative to the current project. That is, if the current project is Base∫, Base∫Sources cannot be denoted by ∫Sources∫.

When a project is created, what one actually creates is a directory whose name is the project name. This directory always contains two files, one called *__CurUserName* that is invisible to the finder but shows up in some dialogue windows, and one called *ProjectorDB* that contains all of the project data. If the project has subprojects, then this directory will contain subdirectories (folders) that similarly house the subprojects. That is, a subproject folder will be named after the subproject, and will contain its own *ProjectorDB* file for the subproject data.

The act of *checking out* a file to a given directory is merely that of "copying" the file from the Projector Database to the directory in question. "Copying" is in quotes because in actuality the projector software may be synthesizing the file dynamically from a set of differential revision data. A checked out file contains a resource named *ckid* which identifies it as a file produced by Projector. This resource contains, amount other things, the file's revision number, whether or not the file is write-protected, and the text of the revision information.

Files may be checked out as *read-only* or as *modifiable*. If the Projector Database is accessible to multiple users, e.g. on a server, then many users may simultaneously check a file out, but only one user at a time may check it out as modifiable.

A file which has been checked out as modifiable may be *checked in* after modification. This enters the modified text as a new revision of that file in the Projector Database.

A *checkout directory* is a directory that has been associated with a project (or subproject) by execution of the command CheckOutDir. This association is private to the user and vanishes when the current MPW session ends. It may also be modified during an MPW session. The association defines the directory into which the files of the project will be checked out by default. There are no restrictions on this association. The seven projects shown in Fig. 1 may be checked out to seven different directories that bear no relationship to each other, may all be checked out to one directory, or (most commonly) may be checked out respectively to a directory structure that matches that of the project hierarchy.

A *name* is an identifier that can be attached to a group of specific file revisions, but only to one revision for any given file. It is used in commands as an pseudonym for this group, most commonly for rapid selection of the revisions belonging to a particular release. Names may be *private* or *public*. Public names become project attributes and are automatically available to all users. Private names are available only to the user who defines them, and last only for the duration of the MPW session.

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

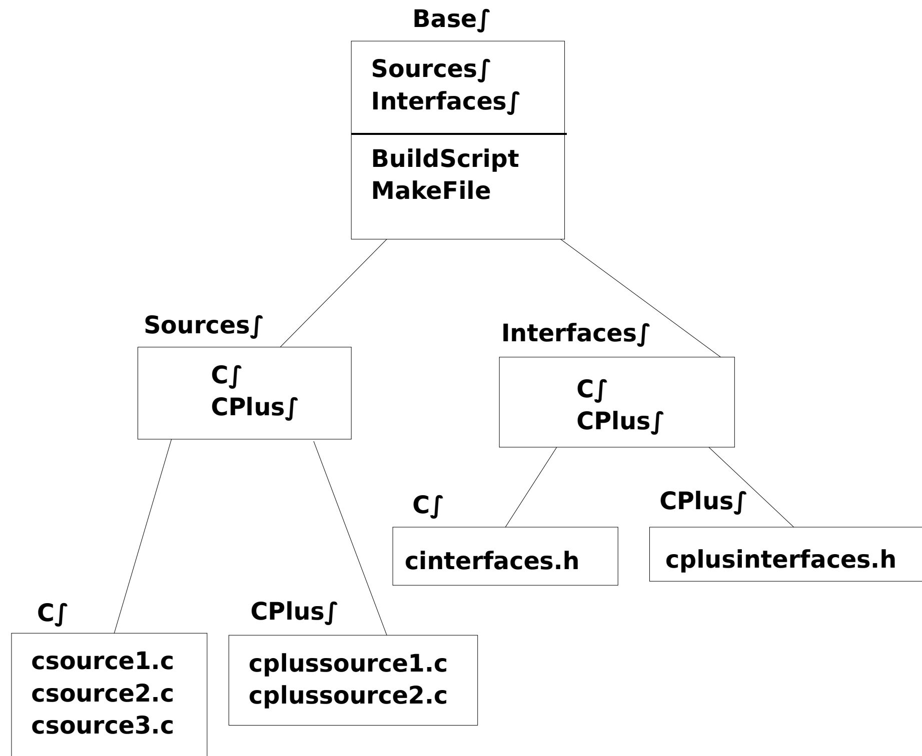


Fig. 1

Project Creation

Let us construct the project hierarchy of Fig. 1. Under the menu item "Project" in the MPW menu bar we can select the item "New Project." The "Project Menu" is illustrated in Fig. 2.

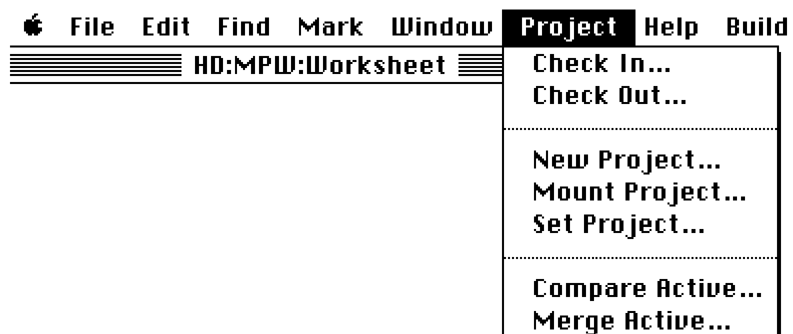


Fig. 2

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

We obtain a special Projector window. It is a bit like a Commando window, but it stays open until one clicks on the “close” box and it can be moved about on the screen. On the left side is a typical file

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

selection sub-window. We wish to establish this project inside of an already-existing directory called “ProjectDemo,” and therefore navigate in the standard way until this directory is selected, giving the window as illustrated in Fig. 3. You can see it already contains two directories, *Documents* and *Examples*, which have nothing to do with the proposed project.. The user enters the name “Base” and the comment respectively into the boxes labelled “Project Name” and “New Project comment.” The User field is automatically set to the value of the MPW variable “user.”

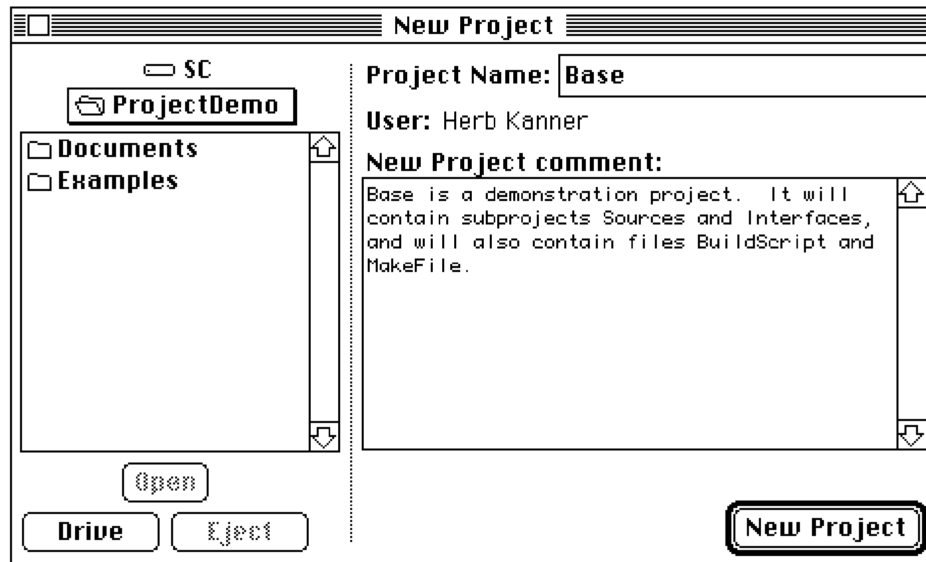


Fig. 3

After pressing the “New Project” button, the window looks like Fig. 4. The directory containing the empty database for the project Base has been created; its name can be seen in the left-hand window.

Now, clicking on the item “Base” in the left-hand sub-window will activate the “Open” button. Clicking on the latter or double-clicking on “Base” will make it the current directory, and now the new projects “Sources” and “Interfaces” can be created as subprojects of “Base” in the same way as “Base” was created. Fig. 5 shows the window after this has been done. The Projector files belonging to “Base,” namely “__CurUserName” and “ProjectorDB” are visible but dimmed. Similarly, the two subprojects “C” and “CPlus”that are subprojects of both Interfaces and Sources can be created. Fig. 6 is the Finder window for Base, showing the folders (directories) for the subprojects Source and Interfaces and the actual Projector file for Base: ProjectorDB. Fig. 7 contains the Finder windows for both Sources and Interfaces, showing their ProjectorDB files and their subprojects.

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

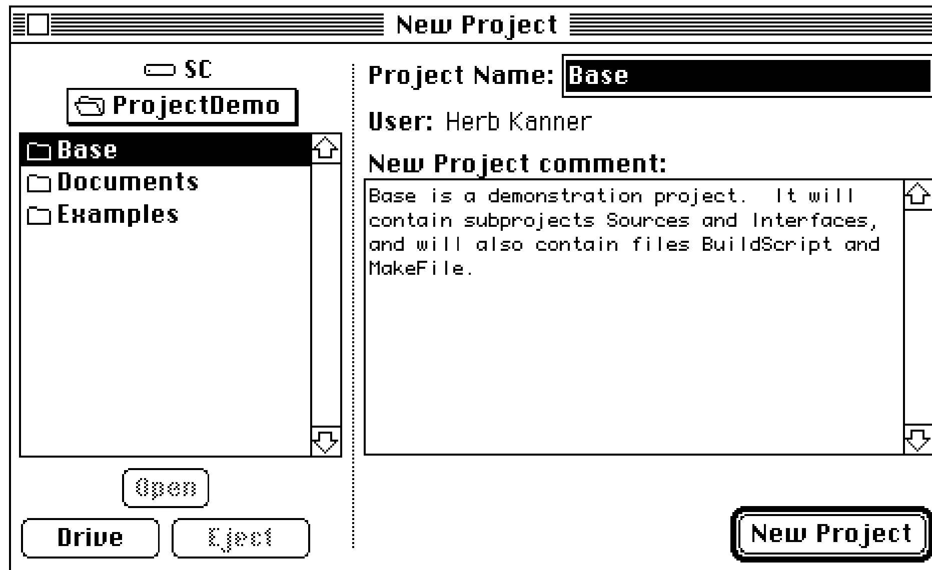


Fig. 4

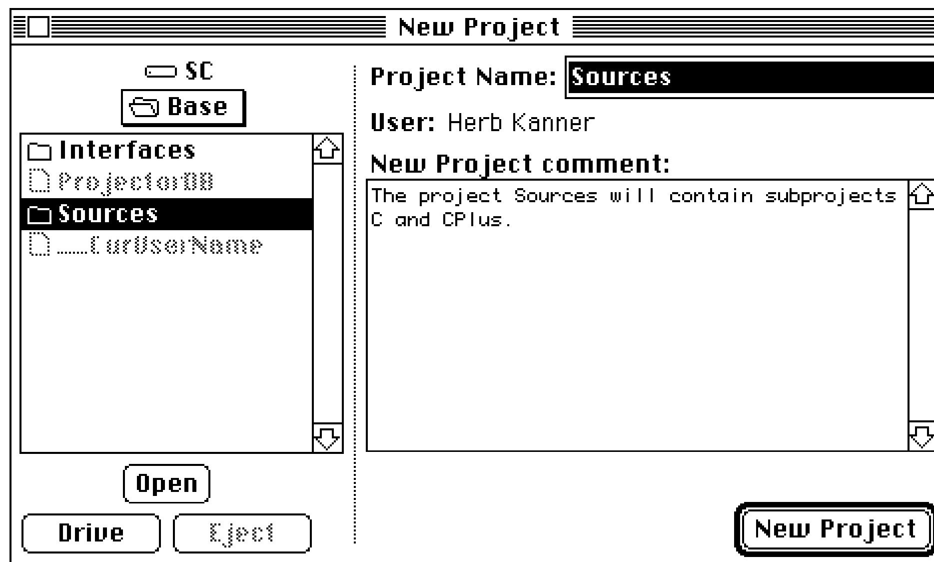


Fig. 5

  Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

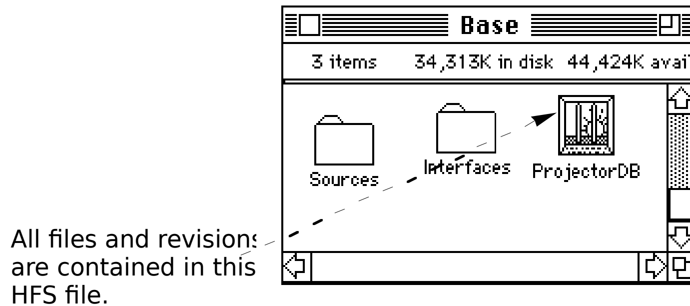


Fig. 6

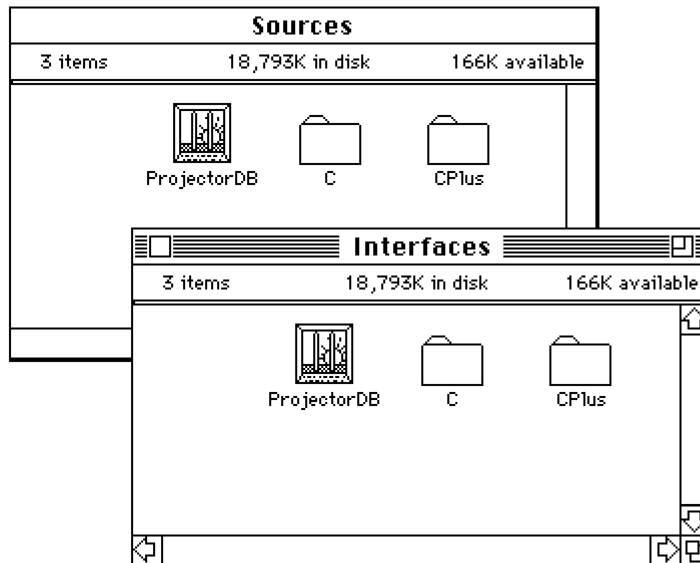


Fig. 7

Before proceeding to the use of Projector for the storage, updating, and retrieval of files, it is necessary to discuss the commands MountProject and Project. What we have done so far is to create a skeletal Projector database, that is a hierarchical project structure and implicitly to *mount* all of the projects. Mounting a project is analogous to opening a file. It makes the MPW Shell aware of the project and makes the project data available to the user. Prior to mounting, projects exist only in file storage media. The next time MPW is initiated, the projects whose creation was illustrated in Figs. 1–4 will have to be mounted again before they can be accessed. The simplest way to do this is with an unadorned MountProject command whose parameter is the project's directory:

```
MountProject SC:ProjectDemo:Base
```

Mounting a project automatically mounts all of its subprojects, so only this one command is required. One can also use Commando, and in fact the Commando window for MountProject can be activated directly from the "Project" item in the MPW menu bar. The radio button should be left at its default value "Generate MountProject Commands." When the button labelled "Project Location" is pushed, the choice "Select a project to be mounted..."

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

will cause generation of the normal dialogue window for directory selection. The one selected will become the argument for the MountProject command.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

A related command that is worth introducing at this point is Project. Analogous to the file system concept “current directory” is the Projector concept “current project.” The Project command is used to set the current project, and, again in analogy to the behavior of Directory, when given with no parameters it returns the value of the current project. A special Projector window is available from the menu bar, called “Set Project... .” It displays all mounted projects in project hierarchy notation (using ∫); the selected one will become the current project.

The MountProject command can also be given with no parameters. In that event, it returns the names of all currently mounted “root level” projects. The default behavior when doing this is to return each name in the form of a complete MountProject command, i.e., to precede the project name with the word MountProject. Figs. 8–10 illustrate some uses and variants of the Project and MountProject commands.

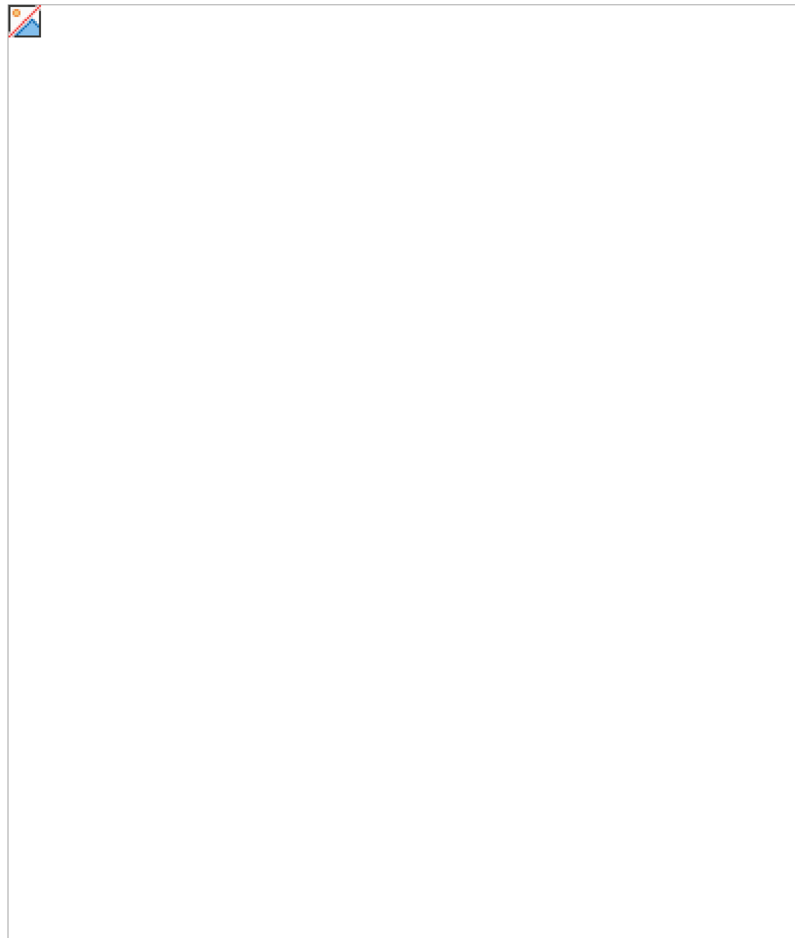


Fig. 8

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

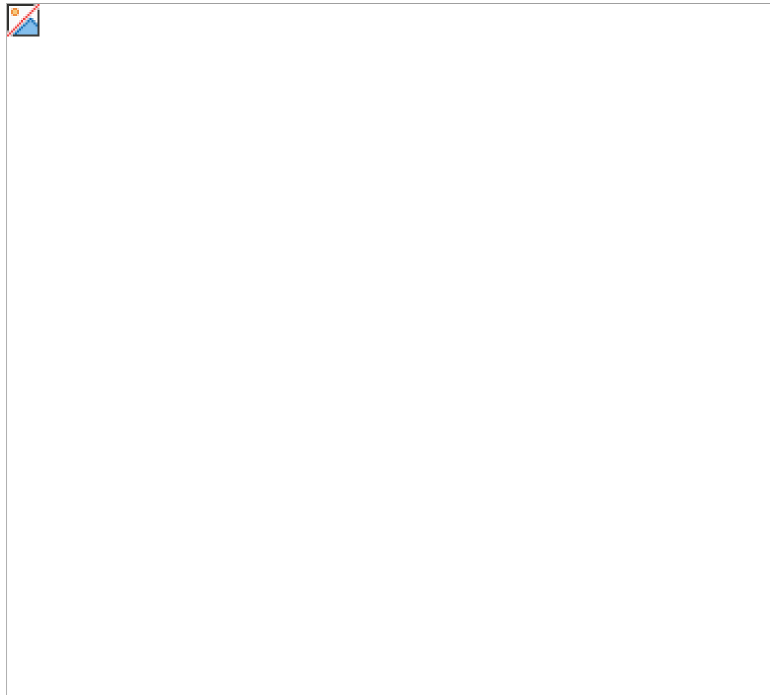


Fig. 9

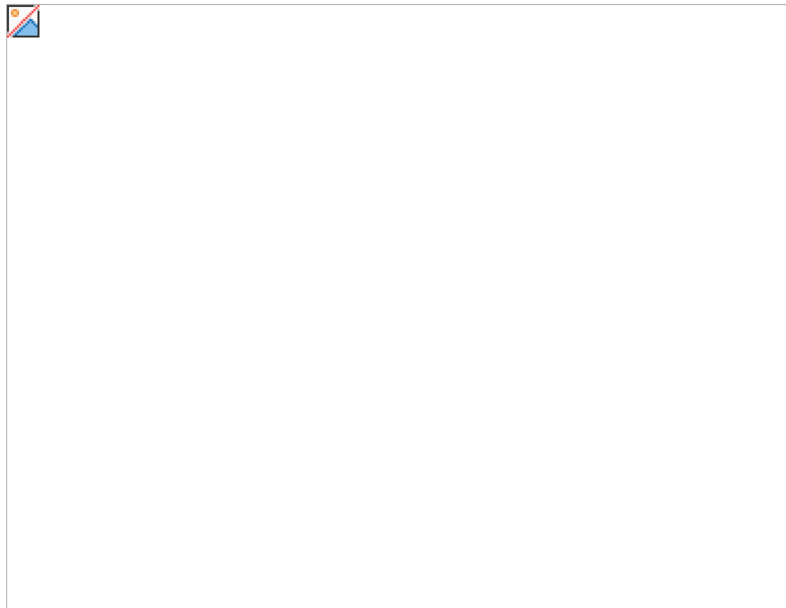


Fig. 10

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

This might be a good moment for the user to experiment with the “Mount Project” and ”Set Project” items in the Project Menu. Mount Project is a conventional Commando window. Set Project creates a window (Fig. 11) that lists, for selection, all mounted projects. It is used in much the same way as the pop-up list of directories in the MPW Directory Menu.

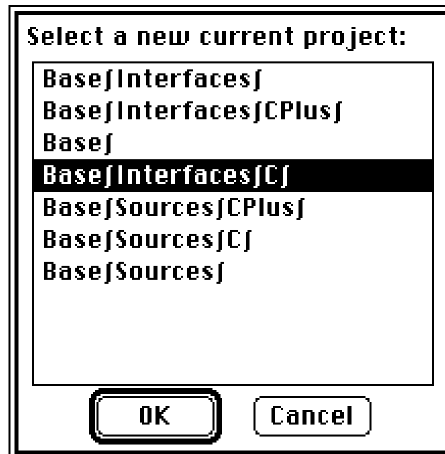


Fig. 11

Relating Directories to Projects

Now that a tree of projects has been created, we wish to put some files into them. Let us make a simplifying assumption which corresponds to the most probably desired organization: that the directory structure into which the working copies of files are to go should be an exact replica of the project structure just created. The first step is to create the directory structure that will in time house the files when they have been checked out of a project. This is done with a command called CheckOutDir. In its simplest form it takes two parameters, a project and a directory. The effect of executing this command is a bit modal: it sets a default so that subsequent CheckOut commands addressed to that project copy the files to the named directory unless another directory is explicitly named in the CheckOut command itself. A side effect of CheckOutDir is that if the directory does not exist, it creates it. A lovely option to this command is **-r**; with this option, sub-directories are created corresponding to all subprojects and they are given the same names as the subprojects. As is the case with MountProject, a CheckOutDir command with no arguments creates an instance of the command showing the directory that corresponds to the current project.

For the purposes of this tutorial, we want to create a set of checkout directories that parallels the project Base. We would like to put them in the same directory that contains Base, namely :ProjecDemo. Since the name “Base” has been already used, we will call the root of the checkout tree “Baseckout.” Fig. 12 illustrates the use of the recursive CheckOutDir command.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

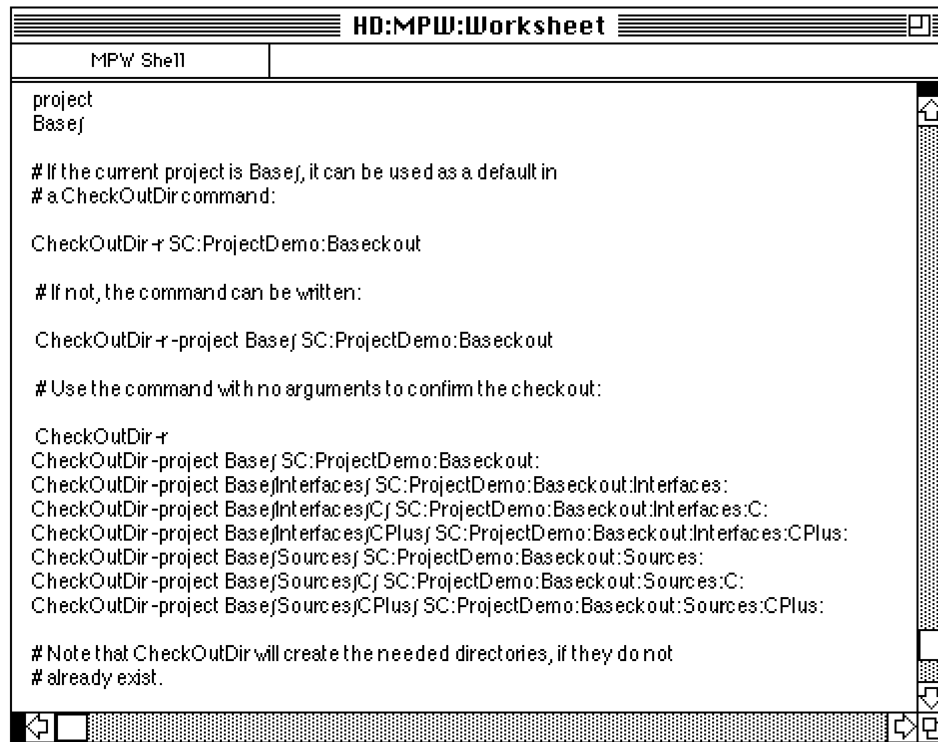


Fig. 12

Checking Files In and Out

The most used and most complicated windows specific to Projector are Check In and Check Out. Check In moves file data from normal file storage into the Projector database. Check Out copies files from the database to normal file storage. The MPW commands CheckIn and CheckOut serve the same purpose, but other than for usage inside of scripts, it is strongly recommended that the windows, selectable by choosing respectively “Check In...” and “Check Out...” in the Project item of the MPW menu bar, be used. These windows remain open until explicitly closed, and can be moved to where desired on the screen. The two windows are partially keyed to each other in that changing the current project on either one affects both windows. Most of the illustrations in this section show both windows, although in practice usually only one is opened at a time. The examples reflect the directory/project structure created in the previous chapter.

Let us now assume that the required files have been written in the appropriate directories, as shown in the following list:—

```
:Baseckout:
        MakeFile
        BuildScript
:Baseckout:Sources:C:
        CSource1.c
        CSource2.c
        CSource3.c
:Baseckout:Sources:CPlus:
        CPlusSource1.c
```

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

  Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

:Baseckout:Interfaces:CPlus:
CPlusInterfaces.h

Opening the Check In and Check Out windows, we get the display shown in Fig. 13.

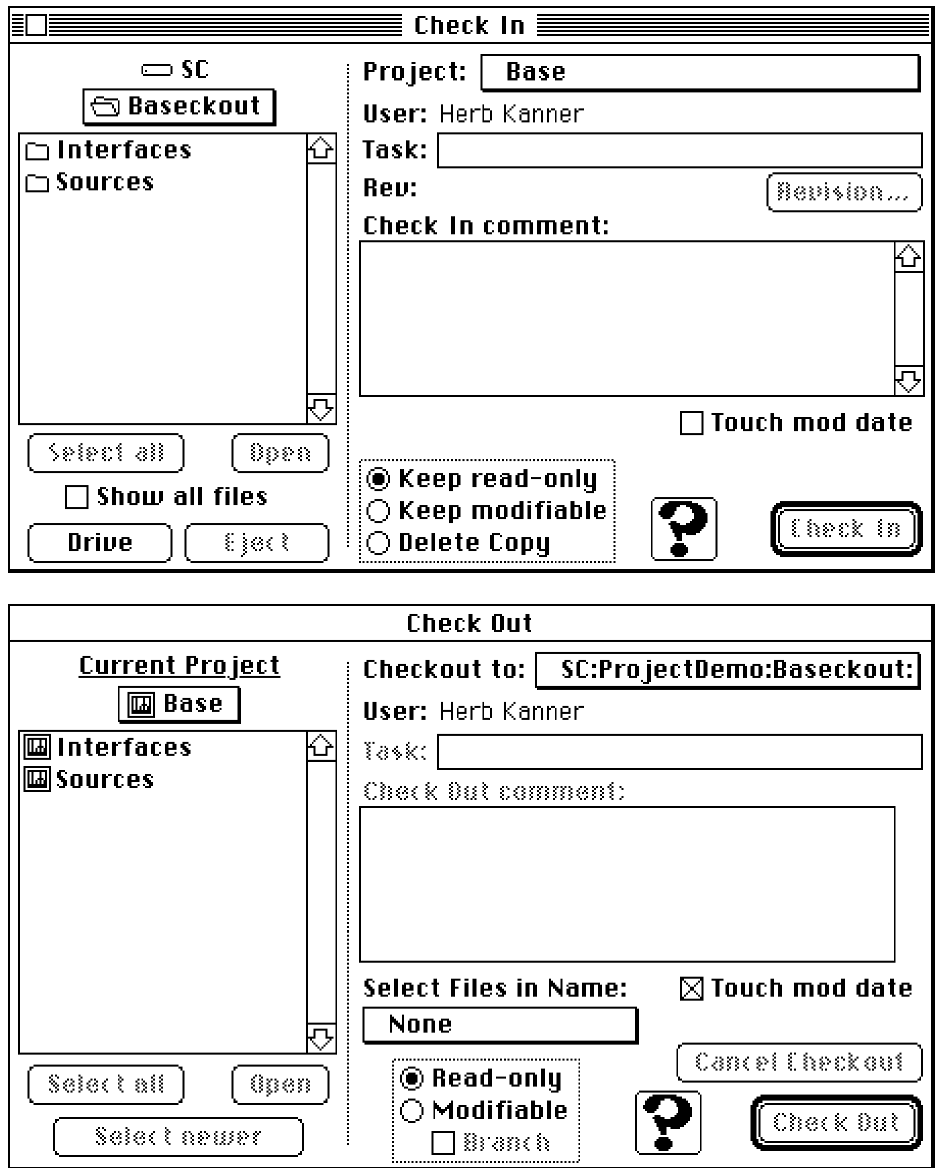


Fig. 13

At the upper right in the Check In window can be seen a button labelled “Project:”. The text on the button is the name of the current project. If no project has been mounted, this text will read “Root level projects.” The current project in this case is Base. If the button is pressed, the names of all projects pop up in a selectable list, with the subproject nesting indicated by levels of indentation. On the left is a display similar to the familiar dialogue used for opening files. Until a CheckOutDir command has been given, it merely displays the contents of the current directory. If the project shown in the “Project:” button has a checkout directory, then that directory will

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

automatically be the subject of the display on the left. The

user may navigate freely within this left-hand portion of the window and select any directory and file on any mounted volume; we will return to the application of this later. On closing and re-opening the Check In window, or even by just leaving and project Base and then returning to it by use of the button at the upper right, the left-hand display will again be set to the checkout directory for Base.

Note that the Check Out and Check In windows track each other in that they both always show the same current project. The left hand display of the Check Out window permits navigation through the project hierarchy and selection of files belonging to the Projector database in exactly the same manner as the customary dialogs for Open do for directories and files. Whatever project is selected as the current project also shows up on the Project button in the Check In window and vice versa. Whatever directory appears automatically near the upper left of the Check In window (right under the name of the volume) will also appear on the button labelled "Checkout to:" at the upper right of the Check Out window. The directory can be changed independently in either of these windows. It automatically reverts to the one established by the CheckOutDir command (if such a one exists) on deselecting and reselecting the project.

Now, let us redirect our attention to the Check In window of Fig. 13. We see in the left-hand display two subdirectories of Baseckout: Interfaces and Sources. No file names are visible even though the files BuildScript and MakeFile reside in the directory Baseckout. The reason is that the Check In window by default only lists files belonging to the current project. Since the project "Base" is brand new, it does not yet contain any files. To make the file names Interfaces and Sources visible in the window, we mark the box labelled "Show all files," yielding the window shown in Fig. 14.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

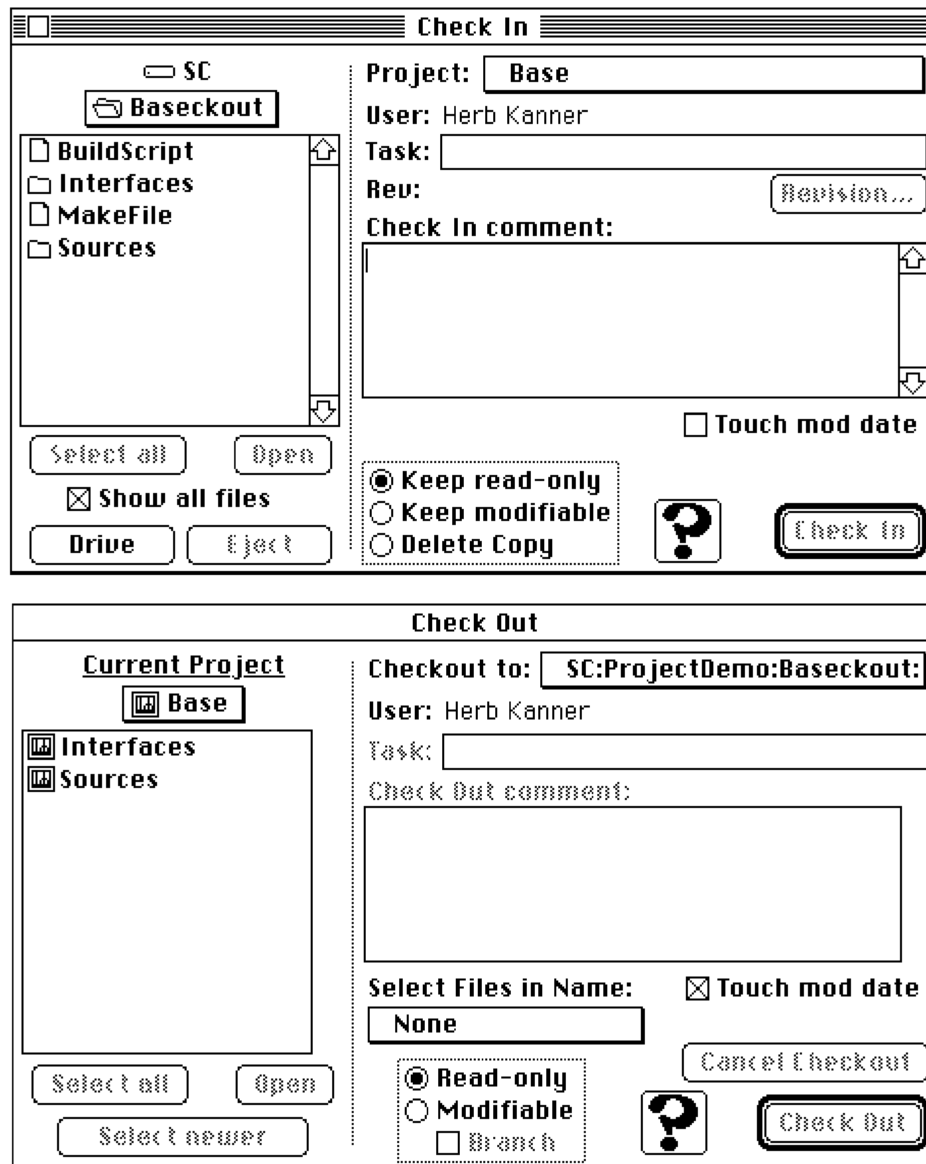


Fig. 14

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

The next step, of course, is to check these files in. Selecting a file and pushing the Check In button will enter the file and all ancillary information about it into the Projector database; this includes any Check In comment that has been written. After this has been done for both BuildScript and MakeFile, the windows will have the appearance of Fig. 15. This illustration also shows the Project pop-up list in the Check In window.

Observe the icons used in both of these windows for files, directories, and projects. For a detailed list of the meanings of all of the Projector icons, see the MPW Reference Manual. Because the radio buttons in the Check In window were left at their default setting (Keep read-only), the names of the two files are now gray, indicating that because they were checked in, and because the copies in the directory Baseckout are now read-only, they clearly should not be checked in again. The icon indicates that they are read-only, and if one now opens the copy of the file in the checkout directory, one will see that same icon in the upper left-hand corner of the file window, and indeed, any attempt to write to that file will meet with failure.

Notice that each window has a button labelled with a great big question mark. Pushing this button changes the right side of the window to an information window and modifies some of the other buttons. Doing this to both windows yields the pair shown in Fig. 16. Note that the window titles have been changed to remind one that they are now yielding information. The same button, now labelled “Done,” when pushed again causes the window to revert to its original state.

Selecting and “opening” a file in the Check Out or Check Out/Information window demonstrates the next level of access, that of the file revision. Fig. 17 shows the appearance of the Information windows after opening MakeFile. We see an icon labelled “1,” the revision number of the only existing revision. As the file is checked out for modification and checked back in, additional such icons will appear, corresponding to all existing revisions. This permits easy fetching of earlier revisions when desired. The radio buttons labelled “Latest Revision Info” and “File Info” permit respectively the choice of a comment that applies to the specific revision selected and a comment that applies in common to all revisions of the file. The term “latest” in the label is not exactly accurate. It reflects the fact that the default selection will, in fact, be the latest revision.

At the start of this section, it was assumed that the original revisions of the files to be checked in already exist in their correct checkout directories. This assumption was made for convenience. By first setting the desired project in the box at the upper right in the Check In window, checking the “Show all files” box, and then selecting desired files for that project by use of the left-hand sub-window, it is easy to check in files from anywhere.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

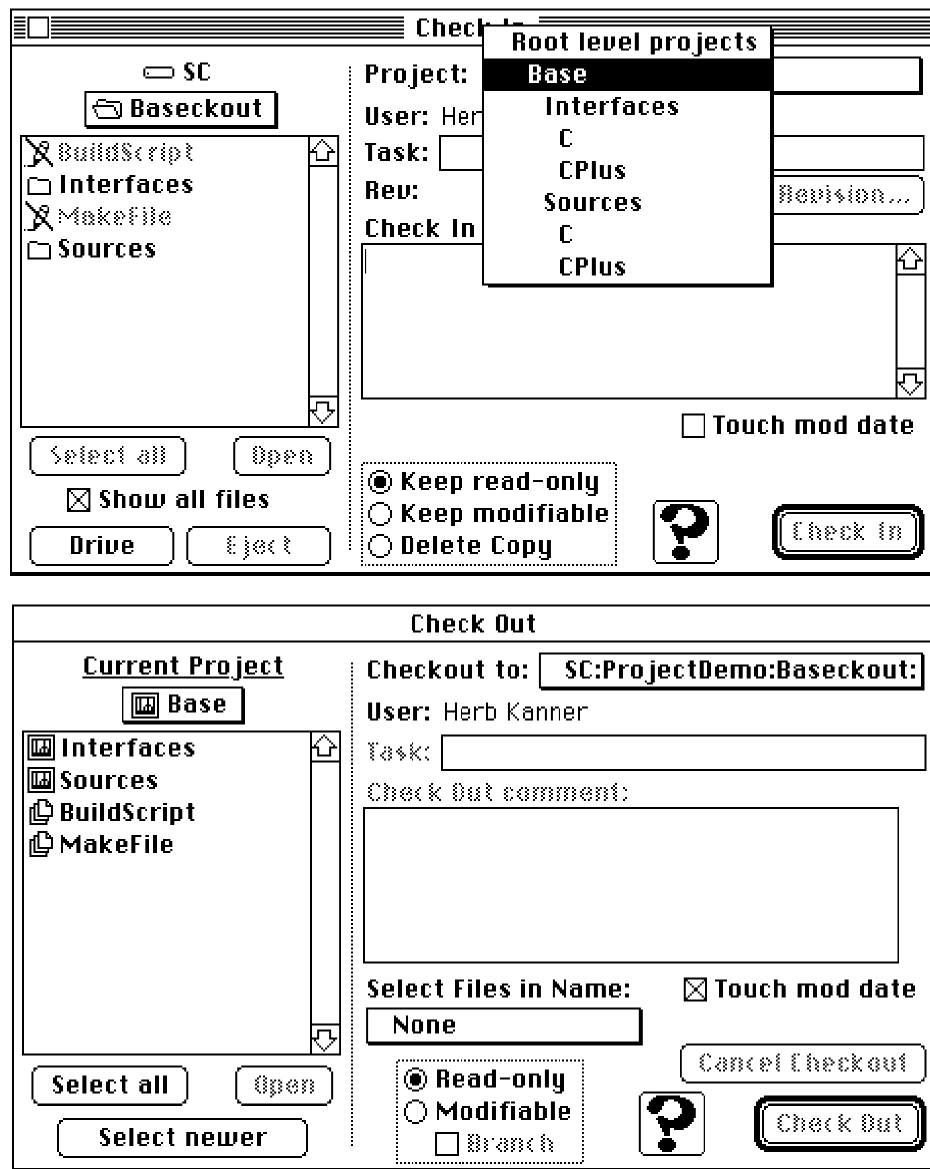


Fig. 15

  Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

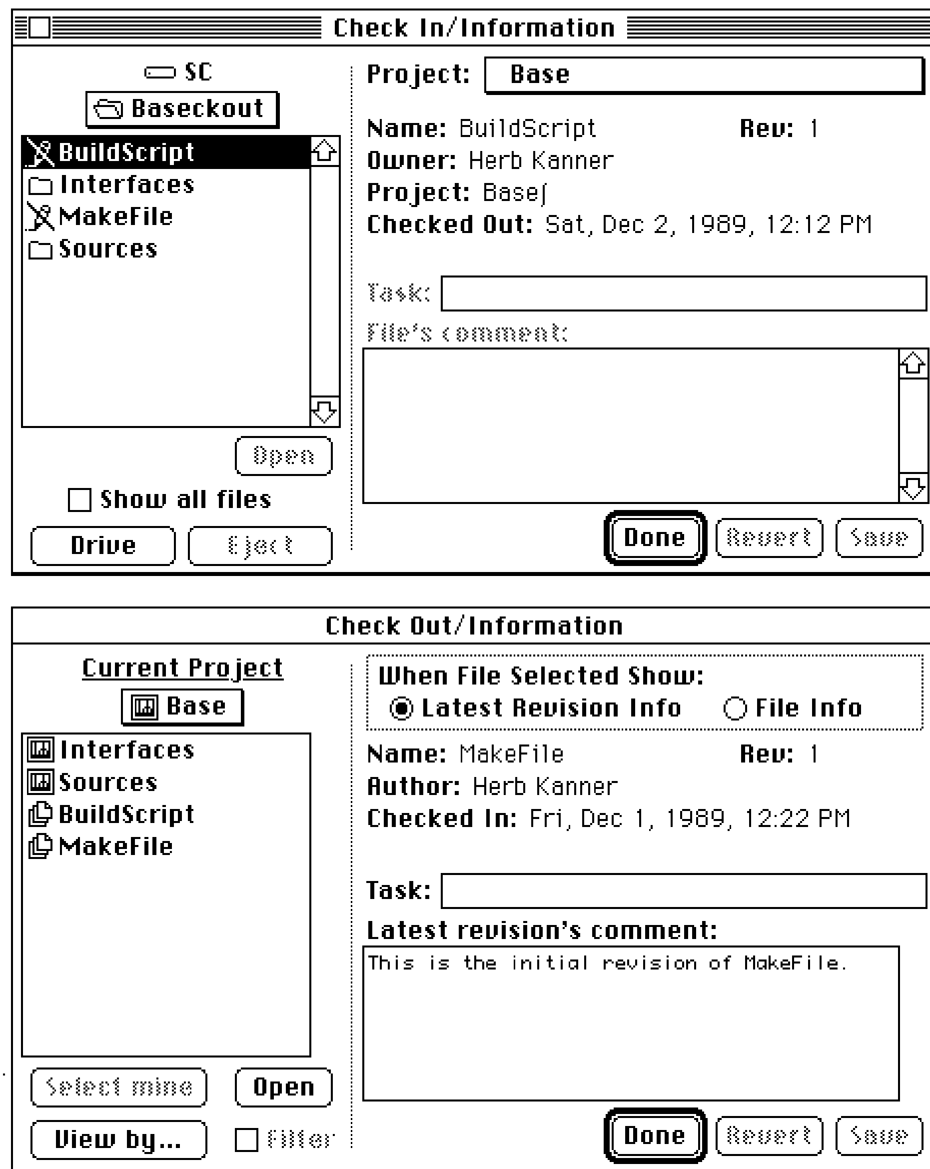


Fig. 16

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

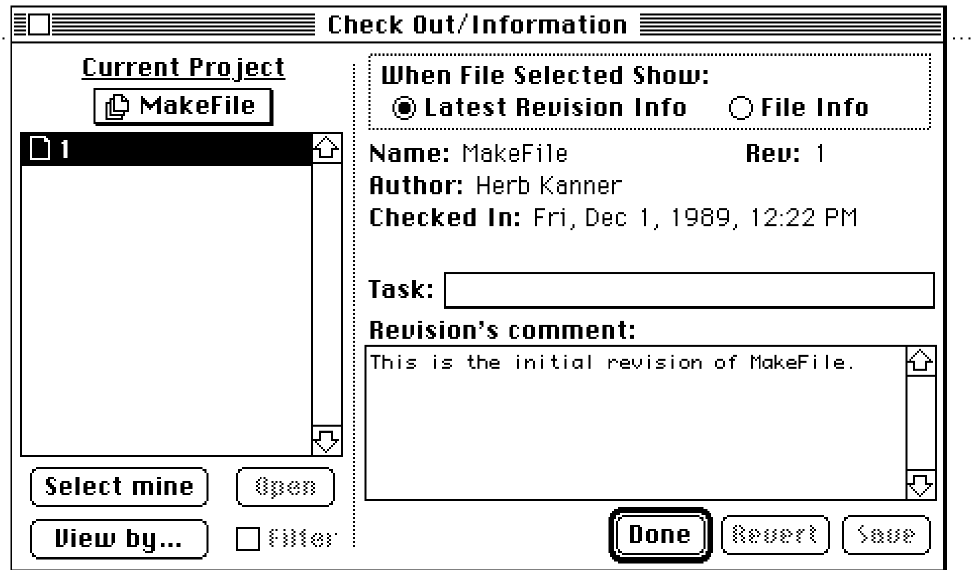


Fig. 17

Let us next assume that original revisions of all the files shown in Fig. 1 have been checked into their proper projects. It is now our intent to make some constructive changes to the file CSource1.c. We go to the Check Out window, navigate in the left-hand area until the current project is Base∫Sources∫C∫, and select the file CSource1.c. Before pushing the Check Out button, we make sure that the radio button “Modifiable” has been pushed. After writing a comment and pushing the Check Out button, we find that the two windows have the appearance of Fig. 18. Note the icon next to CSource1.c in both windows; it indicates that the checked out copy is modifiable. If we now go to the Check Out/information window by pushing the big question mark and open CSource1.c, we get the display of Fig. 19. We see that the revision being modified is called “1+.” After it has been checked back in, this number will change to 2.

This last demonstration could also have been accomplished using the Check Out window, instead of using the Check Out/Information window. However, the file CSource1.c appears dimmed in that window. This is correct. Because it has been checked out for modification, any other attempt at checkout must be only as a branch. Making the file unselectable cautions the user. This precaution can be overridden by holding down the *option* key while clicking on the file name. The file can then be opened to display the revisions as before. Notice that when an “open” is forced on a file that has already been checked out for modification, the box labelled “Branch” in the Check Out window is automatically marked.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

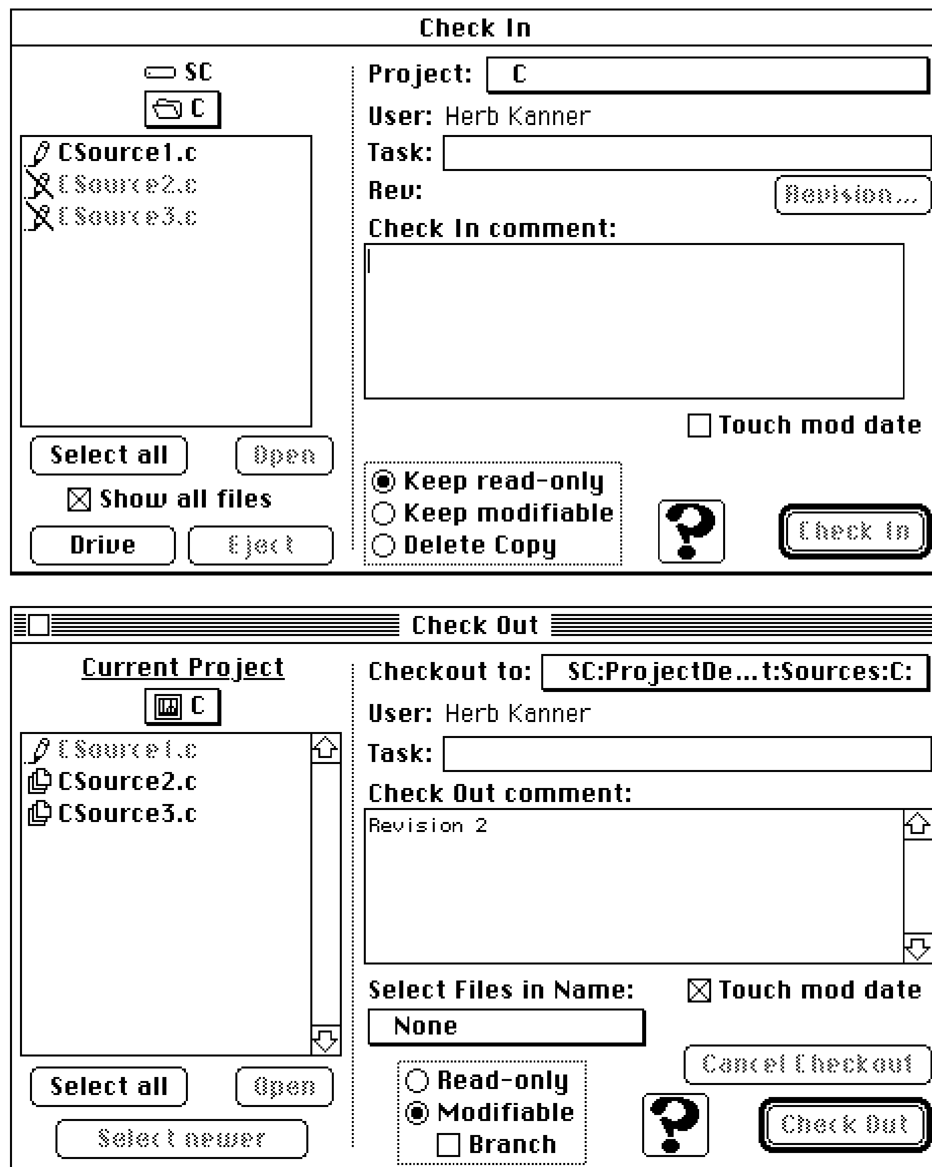


Fig. 18

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

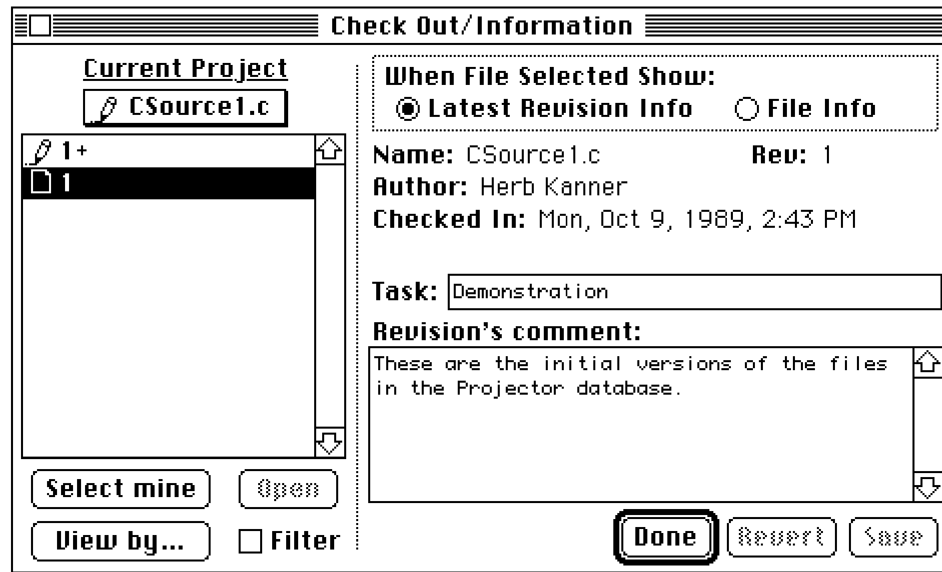


Fig. 19

Branching

The normal sequence of checking out a file as modifiable, editing it, and checking it back in produces what is called “the main trunk,” a series of revisions that are numbered in sequence: 1, 2, 3, ... The button labelled “Revision...” in the Check In window may be used to create gaps in this sequence. That is, if revisions 1 through 4 exist, so that revision 5 would be created next, the use of this button makes it possible to name the next revision with an integer greater than 5. Often, it is desirable to pursue a parallel development while work on the main trunk proceeds. The revisions belonging to the parallel development are said to be on a *branch*. Methods will be shown later for merging files developed along a branch back into the main trunk. The notion of branching is recursive; a branch may be created that diverges from an already existing branch and this may be done to any desired depth. Multiple branches may be taken from the same revision. There is a numbering scheme for branch revisions which enables the user to visualize the tree, knowing only the revision numbers. This branching capability accounts for the term *revision tree* to describe the set of revisions of a file.

Branching from the latest revision is simple. If, for example, the current revision of CSource1.c is Revision 3, then all that is needed is to click on the Branch box before checking out the file as modifiable. The file while it is checked out will be labelled Revision 3a+, and on being checked back in will become revision 3a1. A second parallel branch from the main trunk would be labelled 3b1 after check in. If 3a1 is checked out modifiable, revised, and checked back in, it becomes 3a2. A branch from 3a2 would become, after checkin, 3a2a1, and so on.

Branching from earlier revisions is slightly more complicated. Let us assume again that CSource1.c is up to Revision 3, and that it is desired to revert to Revision 1 and branch from there. Go to the Check Out window, press the “Modifiable” radio button, then select and open CSource1.c. The three revisions will now show, but all but the last will be dimmed. Select Revision 1 by holding down the *option* key while clicking on it. Notice that the Branch box will be checked automatically. An alternative procedure is to select Revision 1 after pushing the “Read-only” radio button, and then pushing the “Modifiable” button. After checkout, the window will taken on the appearance of Fig. 20. After the branch is checked back in, the window will be as shown in Fig. 21.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

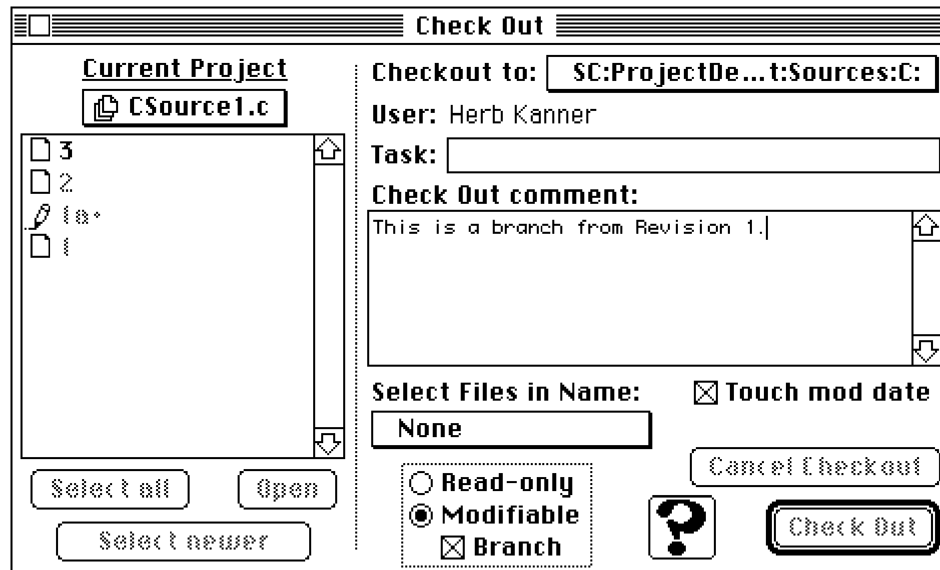


Fig. 20

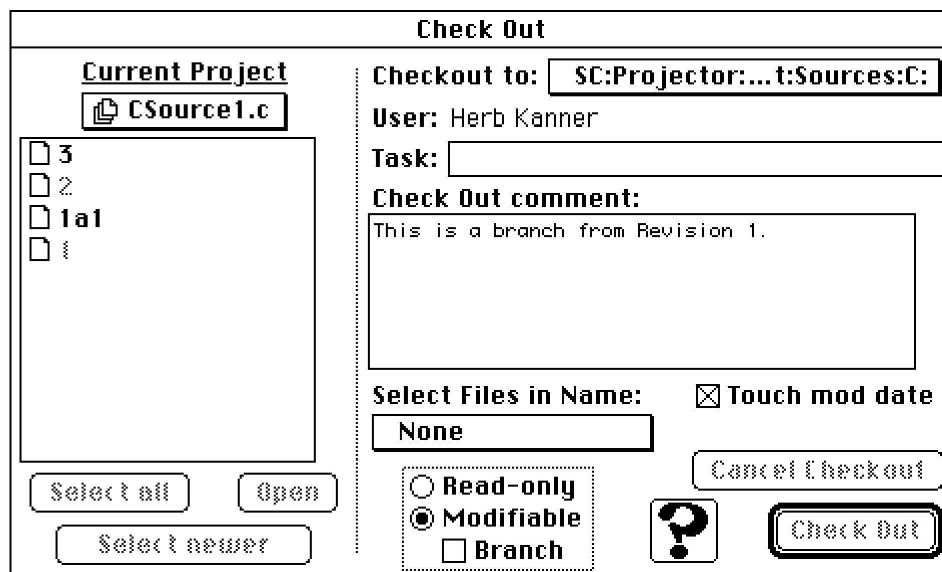


Fig. 21

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

Fig. 22 shows the members of a moderately bushy revision tree (the one hidden item is Revision 1), and Fig. 23 shows the same tree graphically.

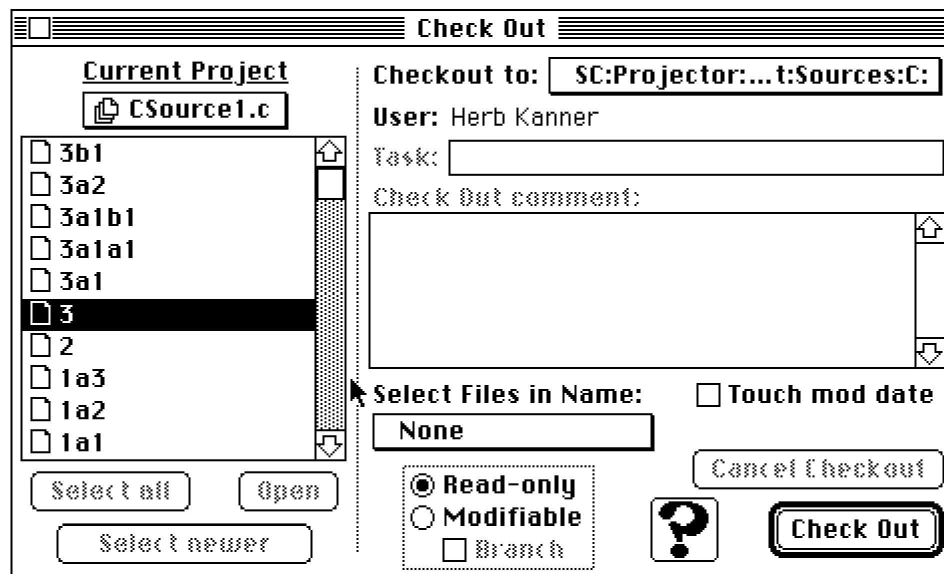


Fig. 22

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

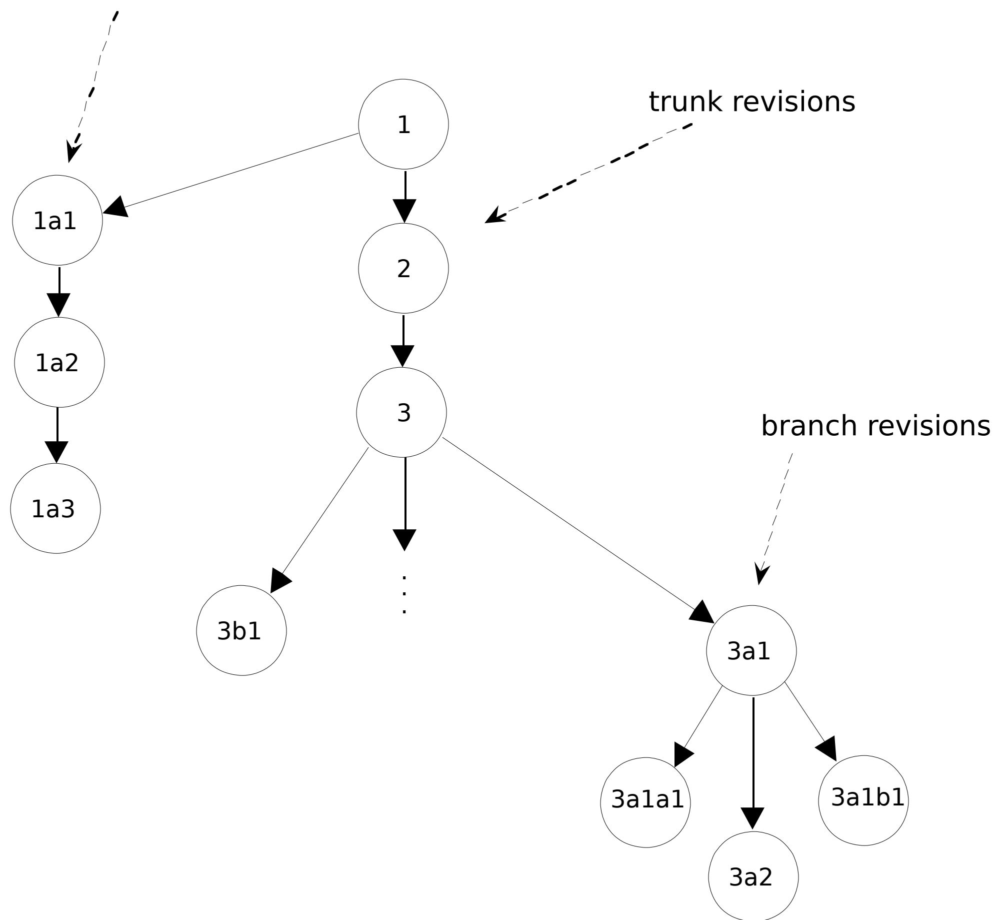


Fig. 23

Miscellaneous Buttons, Icons, and Special Keys

We are now in a position to discuss most of the remaining features of the Check In, Check Out, and New Project windows.

The boxes labelled “Touch mod date” to be found in both the Check In and Check Out windows cause the date of latest modification in the file system directory to be set, respectively, to the time of the checkin or checkout. By default, this is marked in the Check Out window and not in the Check In window. Although this can cause unnecessary revisions of this date, it guarantees an update on every checkout, meaning that tools like Make will always assume that they are being presented with a new version. If this default is not used, and more than one person is working on a file, then there is a danger that a user may check out a revised file and send it to Make without the latter program realizing that the file has been updated. If there is only one user working with a set of projects, reversing this convention and touching the mod date on checkin may be more convenient.

The button labelled “Cancel checkout” is active when a file that has been checked out for modification is selected. Pushing this button changes the status of the file to read-only and discards any changes that had been made to the file while it was modifiable.

The button labelled “Select all” causes selection of the latest revision on the main trunk of all files in the current project. It does not perform a checkout, just a selection. The button labelled “Select newer” selects, with one exception, those files for which the newest main trunk revision is not already to be found

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

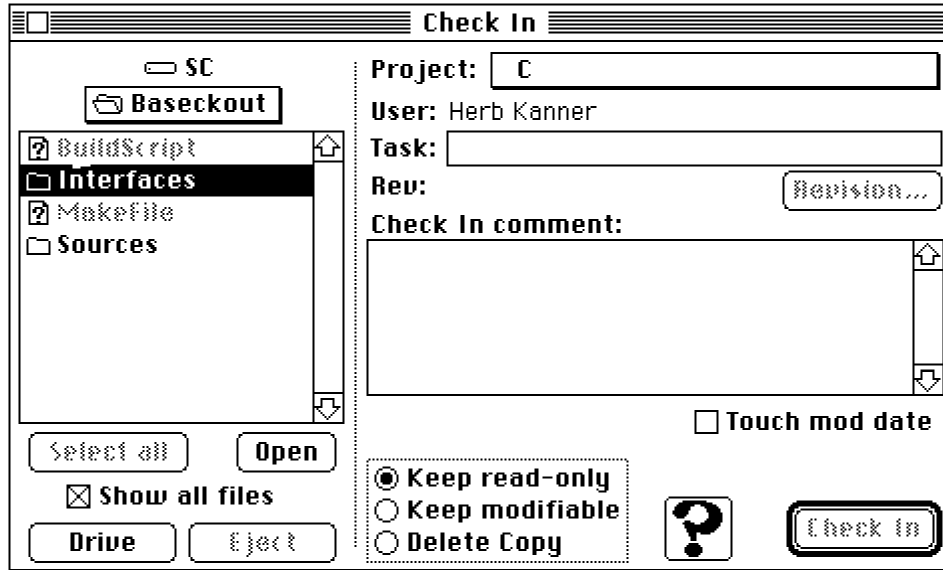
in the user's checkout directory. The exception is any revision which is on a branch. The assumption is that if a branch has been checked out, the user intends to keep it. This button does not distinguish between the main portion of a branch and sub-branches. A revision is either on the main trunk, i.e. its revision number contains no alphabetic characters, or it is on a branch, i.e. its revision number contains one or more alphabetics. If the *option* key is depressed while the "Select Newer" button is pressed, the selection action is modified so as not to select any revision whatsoever of a file unless a copy of the file already exists in the user's checkout directory. This is equivalent to using a written CheckOut command with the **-update** option. The idea is: select file revisions for checkout by the same criterion as "Select newer," but do not check out revisions of any files that have not already been checked out. Just update the files that the user has checked out.

Multiple selection of a subset of the files shown in a Check In or Check Out window can be done in two ways. If the *shift* key is depressed while a second selection is made, then the previous selection is retained and all intervening names are selected, i.e., a contiguous set of names is selected. If disjoint set of names names are desired, then the *command* key should be depressed while making the selections after the first one. The "action" button of the Check In, Check Out, and New Project windows is keyboard activated by pressing the *enter* key. This is required because keystrokes, including that from the *return* key, are sent to the comment field in all three of these windows.

Depressing the *option* key while pushing the Check Out button will cause automatic opening of the file being checked out if it is a text file.

Some less frequently encountered icons are illustrated in the next two figures. Suppose that the user manually selects a directory in a Check In window that is not the "checkout directory" for the current project. For example, this might be done if the directory contains a file which is not yet a Projector file, and the user wants to check this file in to the current project. On marking the "Show all files" box, any Projector files in the directory which do not belong to the current project will be designated by an icon bearing a question mark and will have their names dimmed. This is illustrated in Fig. 24, where the current project is shown as C (actually Base∫Sources∫C) but the user has selected the directory Baseckout. Fig. 25 illustrates one other icon that may be seen when multiple users have access to a Projector database. The user "joe" wishes to check out the file CInterfaces.h. The padlock icon indicates that this file has been checked out for modification by another user. Projector will only allow "joe" to check out the latest revision as a read-only file. If "joe" wants to do any modification, he will have to create a branch.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.



 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

Fig. 24

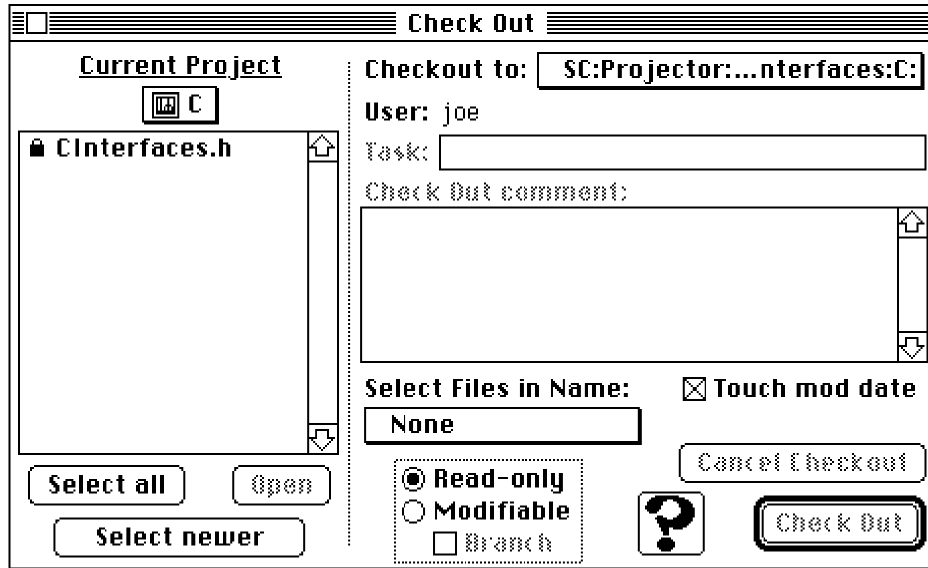


Fig. 25

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

Finally, the Check Out/Information window has a facility for identifying the revision of a file that is currently checked out. The procedure is as follows: Obtain the Information window by pushing the button with the big question mark. In the left-hand area of the window, select and open the file in question thus showing its revision tree. Push the "Select mine" button. If any revision of that file exists in its checkout directory, that revision will be selected in the window, thus giving the desired information.

Naming a Set of Revisions

Facilities exist in Projector for associating a name with a chosen set of file revisions. Thus, for example, the revisions corresponding to a given release, say **alpha1**, of a product can be given that name. As is illustrated in Fig. 26, the button in the Check Out window labelled "Select Files in Name:" will, when pushed, display a pop-up list of all known names. Dragging to the name "alpha1" will then cause selection of that set of revisions, enabling easy checkout of the source files for that release.

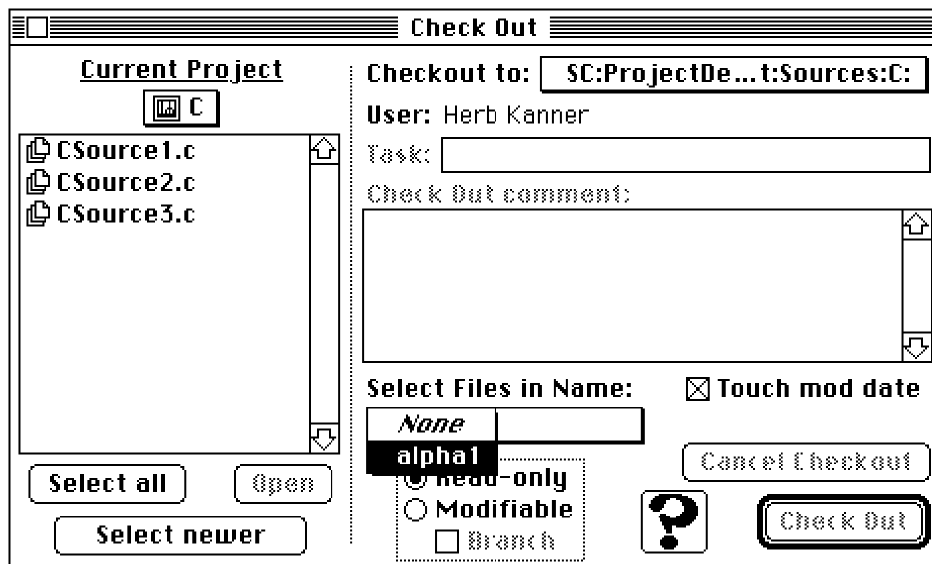


Fig. 26

The assigning of names is done with the command NameRevisions. Like several other commands already described, this command is used with parameters to associate a name with a set of revisions, and without any parameters to elicit information about existing name assignments. It is one of the more complicated commands in the lexicon. Fortunately, anything done with this command can be easily reversed with the command DeleteNames.

The three most important options of the NameRevisions command are **-public**, **-private**, and **-dynamic**. The **-public** option, which is the default, establishes a name as public and relatively permanent. It is recorded in the Projector database and will appear in the Check Out windows of all users. A private name exists only for the convenience of the user who defined it, and lasts only for the duration of the current MPW session. The only way to give it any longevity is to have the command that created it saved in a script file. Private names appear first in the pop-up list and are separated from public names by a dotted line.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

The option **-dynamic** is a little harder to explain. Let us consider a couple of scenarios that illustrate when this option is and is not wanted. First we will describe the simpler case, the one where the option is not used. Suppose that the latest revision on the main trunk of every file in a project is to be used for a release. The NameRevisions command written

```
NameRevisions -project myproject -a thisrelease
```

will freeze the name “thisrelease” to the latest main trunk revisions of files in the project “myproject.” The **-a** option indicates that we want all of the files in the project. The selection is static. That is, at some future time, by which many of the files may have been revised several times, the use of the name “thisrelease” will select that frozen set of revisions. The second scenario might be that one or more files in a project become obsolete. Suppose that a project has files valid1, valid2, valid3, obsolete1, and obsolete2. It has been decided that obsolete1 and obsolete2 will no longer be used. The command

```
NameRevisions -project myproject -dynamic ∂
      active valid1 valid2 valid3
```

will cause the name “active” to be a selector for the latest main trunk revisions of the three named files—that is, the latest revisions at the time the selection is made, not at the time the NameRevisions command was executed. Incidentally, if what was desired was the latest revision on branch “a” of, for example, valid2, that file name would be written in the NameRevisions command as “valid2,1a”. If a revision is fully specified in a file name, e.g. “valid2,1a1”, then that will be the selected revision, regardless of whether or not the dynamic option is used.

If a NameRevisions command reuses a name, the file revisions named in that command will be appended to the list of those previous associated with the name. If the intent is to replace the old set of revisions by a new set, then the option **-replace** must be used. A name can be expunged by using it as an argument of the DeleteNames command.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

Fig. 27

The application of NameRevisions is demonstrated in the next few illustrations. In Fig. 27, using the **-r** (recursive) option, the name "wednesday" is associated with the all files in base∫ and all of its subprojects. Note that two diagnostic messages are emitted about the subprojects that do not contain files. As in MountProject, omission of necessary parameters causes that command to emit information, but includes the command name itself for possible future use. So, using no parameters, the execution of the command with the **-r** option causes printouts of the NameRevisions commands for each level of the project hierarchy that would associate the name with the

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.
applicable revisions. The effect is to get a nice list of the exact revisions that would be selected by use of the name.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

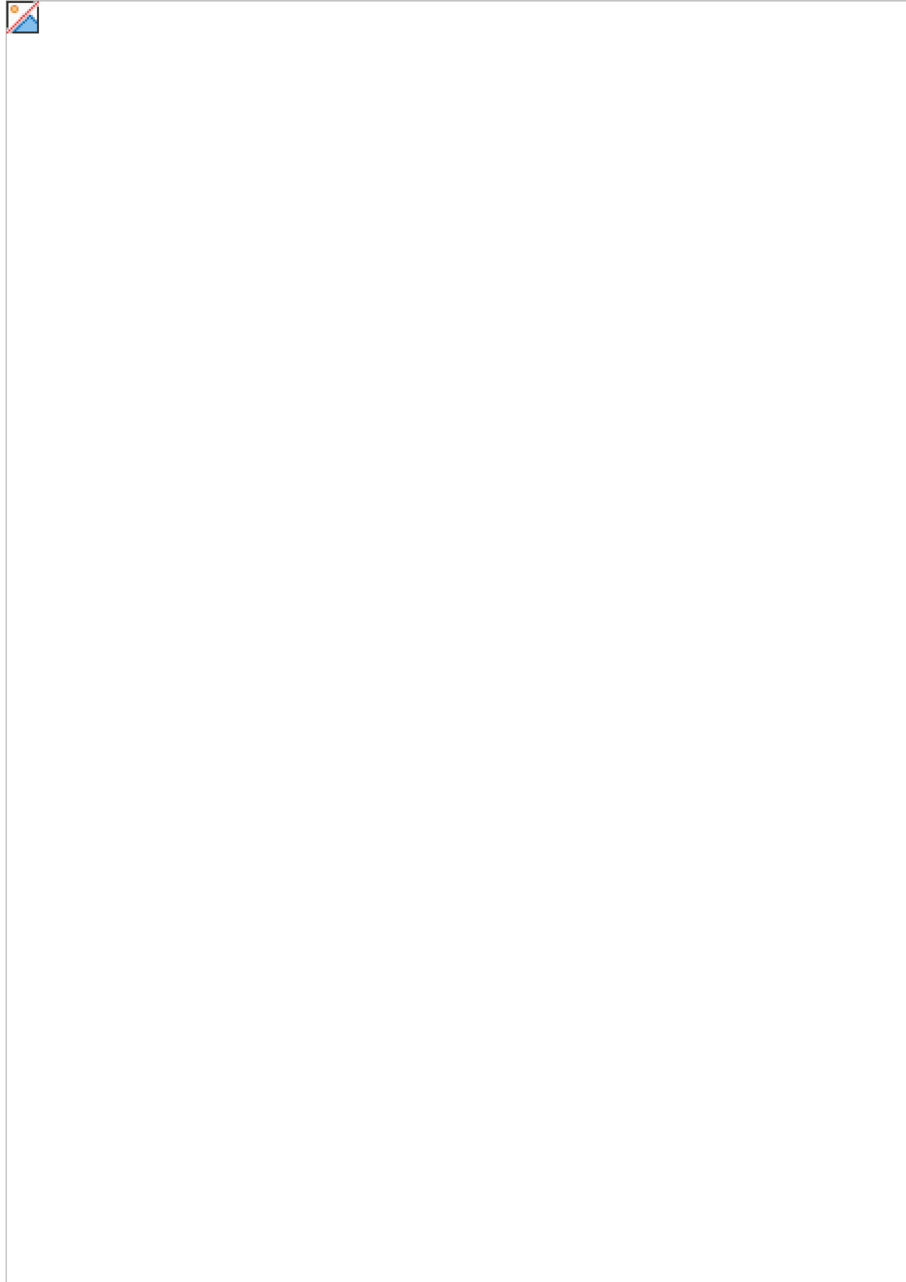


Fig. 28

In Fig. 28, the exercise of Fig. 27 is repeated, but this time the name “latest” is defined to be a dynamic name. The parameterless NameRevisions command is now given an additional option, **-b**, which forces it to list both public and private name assignments. We get to see now the files associated with the public name “latest” and with the private name ”wednesday.” Note that the files associated with the dynamic name are indicated by the file name followed by a comma, but with no revision number. This is because a file selected by a dynamic name is

Copyright Apple Computer, Inc.
1990-1991. All rights reserved.
automatically the latest main trunk revision, so it is not

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

usually desired to see the revision number. If the revision number is wanted, the option **-expand** will force it to appear as is demonstrated in Fig. 29.

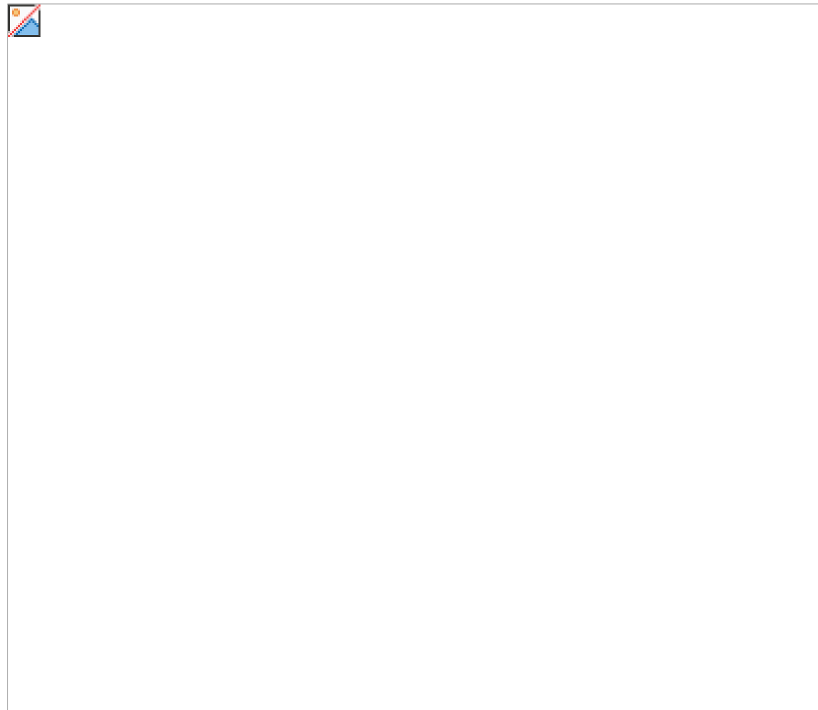


Fig. 29

If You Must Be Different

The demonstrations given so far make the checkout directory mimic the structure of the project itself. Each individual user of the project is free, however, to structure the checked out files in any desired way. For example, the commands:

```
CheckOutDir -project Base∫Sources∫C sc:projector:CFiles
CheckOutDir -project Base∫Interfaces∫C sc:projector:CFiles
CheckOutDir -project Base∫Sources∫CPlus sc:projector:CPlusFiles
CheckOutDir -project Base∫Interfaces∫CPlus sc:projector:CPlusFiles
```

will create directories ":CFiles:" and ":CPlusFiles:", will cause all files in Base∫Sources∫C∫ to be checked out in the directory ":CFiles:", and so on.

Comparing Revisions and Merging Branches

We discuss here the last two items on the MPW menu bar under "Project." The first, named "Compare

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

Active...", calls the script CompareRevisions. The second, named "Merge Active...", calls the script MergeBranch. Both of these scripts do a small amount of housekeeping and call on the MPW script CompareFiles for the main body of the task. When "Compare Active" is invoked—the active window must

  Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

be a checked out projector file—a selector window will appear naming the revision number of the active window and listing all other revisions so that one can be chosen. When the desired revision has been chosen, CompareFiles goes into action, and its windows display all the differences between the two revisions. "Merge Active..." has a similar set of mechanisms and some constraints. The active window must be a branch revision. The other window, implicitly chosen, is the latest main trunk revision, checked out for modification. The mechanisms of CompareFiles permit the user to find the differences and selectively to copy and paste material from the branch to the main trunk. This is the method to be used when work on a branch proves to be fruitful and it is desired to incorporate that work into the main line of the project. For details on the behavior of CompareFiles, see Volume 2 of the MPW Reference Manual.

A particularly simple case of the application of Merge Active is of frequent occurrence, and is worth examining in detail. Suppose we check out Revision 10 of a file for modification, apply a set of changes applied, and then check it back in. After some thought, we decide that the current revision, namely Revision 11, is absolutely worthless and that we would like to revert to the previous revision. Unfortunately, there is no way to delete a single revision from the top. The best that we can do is to create a Revision 12 that is a duplicate of Revision 10. Here is the quickest way to do this. In the Check Out window, “open” the file to show the revision list, chose Modifiable and Branch and then select Revision 10 while holding down the *option* key. Press the Check Out button. A branch revision from 10 will be checked out. Now open the file, so that the active window contains this revision. Select “Merge Active...” from the Project menu. When the machine has settled down, the following will be seen: At the top of the screen will be two windows, side by side. One of them will be read-only, and will be the branch 10a1. The other will be read-write, and will be the modifiable Revision 11+. (Revision 11+ becomes Revision 12 when checked in.) Inspection of the Check Out window will confirm the status of these windows. Now, do a “select all” operation, either via the Edit menu or from the keyboard, on both of these windows, and do a copy and paste of the entire branch window into the modifiable window. Next, select the pop-up item “Done” from the ”Compare” menu item. This will dispose of the two windows. Finally, do a check in of the file, creating Revision 12.

Miscellaneous Goodies

Projector commands not yet considered are:

DeleteRevisions
ModifyReadOnly
OrphanFiles
ProjectInfo
TransferCkid
UnmountProject

These are discussed briefly where the writer feels that some explanation is useful. Some of them are not mentioned at all on the grounds that the material in the MPW manual is adequate.

DeleteRevisions does not do what one might expect. It cannot usually be used to expunge a mistake. Its purpose is to delete large revision sets that are no longer wanted because of obsolescence. So, the alternatives are: with the **-file** option, it will delete all revisions of the named file from a project. It will be as if the file had never been there. Without the file option, it deletes all revisions that are older than the named one on its branch, or an entire branch (see below). The obvious purpose is to get rid of stuff that is so old that no one will ever again want to see it. So, for example,

```
DeleteRevisions -project proj∫ file.c
```

will delete all revisions of file.c prior to the latest one on the main trunk. If the parameter is `file.c,2a3`, then all revisions on branch 2a prior to 2a3 will be deleted. If the parameter is `file.c,2a`, then all of branch 2a will vanish.

ModifyReadOnly is an emergency kind of command. Suppose a file has been checked out read only, and then

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.
the Projector database becomes temporarily unavailable. A typical situation causing this would be that the database is on a server, and the checkout is to a portable medium which the owner takes home. At

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

home, a decision is made to edit this file. ModifyReadonly will remove the read only restriction from the file. This change can be confirmed by that fact that the icon in the lower left corner of the file window changes. The solid line that crosses the pencil becomes a dotted line. When this file next confronts the Projector Check In window, this same modified icon will appear next to the file name, and it will be possible to check the file in as a new revision.

OrphanFiles is used to remove completely the Ckid resource from a file, so that the file is no longer recognized by Projector.

 Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

Command Syntax

CheckIn -w | -close | ([-u *user*] [-project *project*] [-t *task*]
[-p] [-cs *comment* | -cf *file*] [-new | -b] [-m | -delete]
[-touch] [-y | -n | -c] (-a | *file…*))

CheckOut -w | -close | ([-u *user*] [-project *project*]
[[-m] [-b] | -cancel] [-t *task*] [-cs *comment* | -cf *file*]
[-d *directory*] [-r] [-open] [-y | -n | -c] [-p]
[-noTouch] (-update | -newer | -a | *file…*))

CheckOutDir [-project *project* | -m] [-r] [-x | *directory*]

CompareRevisions *file*

DeleteNames [-u *user*] [-project *project*] [-public | -private]
[-r] [*names…* | -a]

DeleteRevisions [-u *user*] [-project *project*] [-file] [-y] *revision…*

MergeBranch *file*

ModifyReadOnly *file…*

MountProject

NameRevisions [-u *user*] [-project *project*] [-public | -private | -b][-
r] [[-only] | *name* [[-expand] [-s] | [-replace]
[-dynamic] [*names…* | -a]]]

NewProject -w | -close | ([-u *user*] [-cs *comment* | -cf *file*]
project)

OrphanFiles *file…*

Project [-q | *projectname*]

ProjectInfo [-project *project*] [-log] [-comments] [-latest] [-f]
[-r] [-s] [-only | -m] [-af *author* | -a *author*]
[-df *dates* | -d *dates*] [-cf *pattern* | -c *pattern*]
[-t *pattern*] [-n *name*] [-update | -newer] [*path…*]

TransferCkid *sourcefile destinationfile*

UnmountProject -a | *project…*