HowToUseAPublicCedarMachine.tioga
Last Edited by: Pavel and Spreitzer on May 12, 1986 1:44:56 pm PDT
Last Edited by: Subhana, May 31, 1984 3:55:01 pm PDT
Last tweaked by Mike Spreitzer on April 27, 1987 4:53:53 pm PDT
Mike Spreitzer June 30, 1986 11:38:29 am PDT

HOW TO USE A PUBLIC CEDAR MACHINE

How To Use A Public Cedar Machine

Release: /Cedar/CedarChest®/Documentation/HowToUseAPublicCedarMachine.tioga

Abstract The peaceful coexistence of several distinct users on a single public Cedar machine requires a certain amount of discipline and courtesy on the part of those users. This memo attempts to address these issues and to lay down guidelines for the use of public Cedar machines. We discuss the True Religion of the local directory facility and how to support, in a politically-correct manner, the differing configurations of files, viewers, etc. as desired by various users.

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

For Internal Xerox Use Only

Introduction, or What's the Problem?

Back in the bad old days of Cedar 4.4 and earlier, before the emergence of local directories, everything any user was working on was put in the same directory on the disk, there being only one. Now, this was fine for the elite many who had private machines, but those downtrodden ones, the swimmers (and sinkers) in the public machine pool, had to continually worry about whether all of their files were still resident, whether the ones with the names they remembered were in fact the ones with the contents they remembered, etc. In the bright new world since Cedar 5, these problems can be solved in a reasonable manner and the second section of this memo describes what we consider the 'correct' way to do so.

There are still difficulties, however, involved with the use of public machines. Individual users have individual opinions about what commands they use, what files they want present and where, how the viewers should be set up, etc. What is needed, and what we provide in the third section of this memo, is a set of conventions and techniques for the setting up of user profiles which gets everyone the machine configuration they want without stepping on each other's toes. Under this plan, it is possible for a user to sit down at a new public machine and have it automatically customized for his or her use.

The memo concludes with some smug claims about how wonderful our suggestions are and the merest hints of recommendations for users of private machines.

How to Sign Up for a Public Machine

At the time of this writing, we don't know how many public machines there will be, but someone can tell you where to find them and their sign up sheets. Once you've found them, you can sign up for a slot of time on them. The rules for doing so politely are as follows:

1) On weekdays between 7:00am and midnight, you may be signed up for a total of no more than 3 hours at any given time. If you use one of your three hours, it's OK to claim another (even contiguous with your remaining time).
2) On weekdays between midnight and 7:00am, and between midnight Friday-Saturday and 7:00am Monday morning, you may sign up for up to eight hours, but *not* contiguous to your daytime sign-up time. For runs longer than eight hours, you should probably talk to somebody with authority around here (if you can figure out who that might be).
3) Sign up as far in advance as you want.
4) You must be the real user – don't sign up under another's name.
5) Hardware and microcode work (neither of which happens very often) take precedence over other uses.
6) If you choose to split up your three hours, segmenting it into chunks of less than an hour is *illegal*, as is bracketing free times of less than an hour. If you sign up for only one chunk, then less than an hour is OK.

The True Religion of Local Directories

Definitions and Some Holy Notation

A hierarchical file system is one in which there is a tree of *directories*, each of which is a repository for files and other directories. They are very handy for organizing things and for limiting the number of files you have to look at at any given time. Of course, the truly enlightened do not distract themselves with directories, but instead use DF-files, which you will find described elsewhere (if you're lucky). Directories also make it much easier to create and delete groups of related files, for example when you are fixing a bug in a piece of public software. The inner circle of cogniscenti are creating and using tools to do similar functions, only in a much better manner, with DF-files.

At any time while running Cedar, each Command Tool has a *current working directory*. This is the directory that will be searched for files whose names you give without the directory part. The current working directory gives you a place of reference, a location within the directory hierarchy. It should be well noted that in the current imperfect world, this working directory is only uniformly supported within the Command Tool. Other viewers have other, usually strange, conventions; you are well advised to stick to full path names within these other viewers.

Cedar directories were inspired by those of UNIX(TM), but have their own special charm and idiosyncrasies. The first thing you may notice about them is that there are two distinct syntaxes for naming them. The older of the two is taken from the TENEX world and looks like this

[Cedar]<Cedar6.1>Documentation>TiogaDoc.tioga

for a remote file and

[]<>Users>Pavel.pa>TiogaDoc.tioga

for a local one. In these examples, 'Cedar' refers to a file server, 'Cedar6.1', 'Documentation', 'Users', and 'Pavel.pa' are directories or subdirectories, and 'TiogaDoc.tioga' is the name of a file. The newer syntax was taken from the UNIX(TM) world and makes these two full path names look like this:

/Cedar/Cedar6.1/Documentation/TiogaDoc.tioga
///Users/Pavel.pa/TiogaDoc.tioga

Different programs in Cedar generate different ones of these two, but nearly all programs will accept both for input.

In both naming schemes, some random set of commands allow you to use the UNIX(TM) convention of the special names '.' and '..'. The first of these refers to the current directory in the pathname, making ///Users/Pavel.pa/./././ the same as simply ///Users/Pavel.pa/, and the second refers to the parent of the current directory, making ///Users/Pavel.pa/../Spreitzer.pa/../../Users/Pavel.pa/ a very long way to say ///Users/Pavel.pa/.

The next most interesting thing about Cedar's directories is that they don't actually "exist" --- that is, they are not reified as files. In fact, they are just a clever way of interpreting file names with embedded '>' characters. It is thus the case that you need not specifically 'create' or 'delete' a directory. If you want, you can think of it as being created automatically when you put a file in it and just as automatically deleted once you've removed everything that was once in it. This scheme has several fun advantages and disadvantages, many of which will probably come to you in due

time if you waste time thinking about it.

Useful Commands for Manipulating Directories

Knowing where you are in the local directory structure and getting to other places involves the use of the following set of commands. *Please note* that while these commands certainly sound like their UNIX(TM) counterparts, some of their semantics are different. For each command listed, its registered abbreviation is given in parentheses. The full documentation on most of these commands is in the file /Cedar/Cedar®/Documentation/CommandToolCommands.tioga.

ChangeWorkingDirectory (CD)

This changes the working directory of the Command Tool in which it is typed to the one specified on the command line. If no argument is given, it defaults to your "home" directory (see next section for a definition of "home"). It is possible (and occasionally even useful) to CD to a remote directory. However, you should be careful about running programs in Command Tools which have a remote working directory; some programs try to create temporary files in the current directory, but FS won't let them do so remotely, so ... Of course, the True Religion states that all temporary files should be created in ///Temp/ on the local machine, but there yet remain infidels. The current working directory of any Command Tool is given in its banner.

PrintWorkingDirectory (PWD)

This prints out the current working directory of the Command Tool in which it is typed. Since the current working directory of any Command Tool is given in its viewer's banner at all times, the only reason we ever use this command is to get the directory name printed out so we can shift-select it into another viewer.

PushWorkingDirectory (Push)

PopWorkingDirectory (Pop)

There is maintained, inside the Command Tool, a stack of working directories. Unless you've used this command, that stack is mightily uninteresting, containing only your current directory. There are times, however, when you are in a directory with a particularly long name and want to briefly visit another directory. Rather than do a CD to the new one and then another, longer CD back again, you can 'Push' to the new directory and just 'Pop' back later. Each 'Push' adds its argument directory to the stack and each 'Pop' takes one off and CDs to the next one. As with CD, Push uses your home directory if given no argument. It is remarkably uninteresting to Pop when the stack is empty.

List (LS)

This lists the files which are contained in the current directory. To limit the amount of output you get or to list the files in some other directory (or both), you can put arguments on the command line in much the same way as in UNIX(TM), using the pattern character '*' as a wild card. You should consult CommandToolCommands.tioga for all of the details, but we'll mention two of them here, since they directly relate to the use of local directories and public machines.

The first niggling point is that if you want to list all of the files in some other directory, you need to state your request a little bit oddly. Suppose I want to list

everything in my home directory. The following command

List ///Users/Pavel.pa

will not, in fact, do what I want. This is a consequence of the fact, mentioned above, that Cedar directories don't really exist. Rather than go into the logic of this consequence here, let us simply point out that if you change it into a real live pattern, List will get it right:

List ///Users/Pavel.pa/*

Our other detail about the use of the List command concerns the '-u' option. This option on the command line tells List to only show those files which have not yet been backed up on a network file server. This is a very handy way to convince yourself that you've got your bags packed when you leave a public machine or wipe your local disk or whatever. As an example, the command

List -u ///Users/Pavel.pa/*!H

will show me every file in my home subtree which has not been copied onto a server and about which I should perhaps do something. If this list is empty, you can be reasonably sure that the Gods could wipe out your disk without harming you. (The "!H" in this command says to only consider the highest-numbered version of each file in making the list.)

PrintSearchRules

SetSearchRules

AddSearchRules

When you issue a command which has not yet been registered with the Command Tool (most any command when executed for the first time), how does the Command Tool find it? The answer is that it maintains a list of directories to be searched for commands in order of precedence. The PrintSearchRules command will print this list for you, SetSearchRules will set it to the list given on the command line, and AddSearchRules will append the list on the command line to the *front* of the Command Tool's list. The Command Tool's list starts out as (///®/Commands/ ///®/System/); you might want to add to it if you have a personal commands directory as discussed below.

OpenR

Strictly speaking, this command doesn't belong here, since it doesn't deal with directories at all, but the policies in this document combine to make OpenR much more important than it's ever been before.

Back in the bad old days, a DF file named LifeSupport.DF, which contained every .mesa and .BCD file for every interface in the release, was BroughtOver into the /// directory whenever a machine was booted. Since this also involved bringing over most of the DF files and documentation files as well, some version of very nearly everything you might want to look at was somewhere in ///. While this had some advantages, such as making it easy to find elements of the release, it was also a pain, since it meant that many hundreds of file names now cluttered up the file system. Thus was born the OpenR command, standing for "Open this file, if it's a member of the release". Thus, if you wanted to look at the source for the Rope interface, you could just say, "OpenR Rope.mesa" and, as if by magic, a Tioga viewer on that file would open up, without you ever needing to know just where that file was on the

remote servers. Obviously, in our improved world with its nice clean /// directory, the OpenR command has become a true friend and important addition to the environment.

The True Religion Itself

This espousing of the True Religion has a number of goals: to make it easy to find things on public machines, to add to users' confidence in the integrity of their assumptions about the local machine, and to have it Our Way. If everyone using public machines follows our guidelines, as laid out in this section and the next, it will make everyone's life a bit easier and the church fathers very happy.

There are four kinds of users of public machines, in relation to the policies laid out in this document: Bad Citizens (people whose actions make it difficult or impossible for others to follow the True Religion), Good Citizens (those who, while not Followers themselves, do nothing to impede the Follower community), Followers of the True Religion (reasonably enlightened souls who use the general policies below), and True Believers (a.k.a. Saints, those who dogmatically follow every tenet of the Church, as laid out in this document). This latter category consists mostly of the authors (at least when anyone's looking).

It is really very easy to avoid being classed as a Bad Citizen. *Thou shalt not* replace with non-standard contents any standard files in system directories (e.g., ///, ///®/System/, ///®/DontDeleteMe/, ///®/Commands/) or any file from any other person or project's home subtree. *Thou shalt not* create any files in any other person or project's home subtree (see below). And finally, *thou shalt not* change the contents of anyone else's profile. Follow these simple rules and it's reasonably likely (no warranty expressed or implied) that no other user will come at you with a Holy Axe.

The first of the rules for Followers is that nothing should be put into the *root directory* (a.k.a. '///' or '[]<>') except those things which the software is currently too stupid to look for anywhere else. At the time of this writing, that list has just recently become satisfyingly small and obscure. Most of these could be moved elsewhere with just a Small Matter of Programming and will be someday. The ideal situation will be realized when the root directory only contains subdirectories.

The various files involved in making commands work out (i.e., the '.load', '.bcd', '.tip', '.icons', etc. files) should be placed in the directory ///®/Commands/. This is the directory in which the Command Tool first expects to find them and it's nice to have them all in one place. Certain other kinds of '.bcd' files which are not, strictly speaking, commands (e.g. Watch.bcd and GetDWIM.bcd) should also go in this directory.

Any program which creates a temporary file during its execution, should create that file in the directory ///Temp/, *not* in the current working directory. The definition of temporary file includes such items as Walnut's various and sundry log files, the log file produced by Chat, and the temporary Press files produced by several programs. It goes without saying, of course, that such files should usually be removed by the program before exiting.

The files for an individual user go in his or her *home directory*. For a given user, their home directory is ///Users/*That-user's-Login-name*/. For example, the home directories of the authors are '///Users/Pavel.pa/' and '///Users/Spreitzer.pa/' respectively. Any files created by an individual should go somewhere in that subtree. That way, everyone can be assured that nobody has stepped on anybody else's toes. Symmetrically, nobody else should put anything into your home subtree: it's your private sandbox. We give some recommendations below about how you might organize your subtree, but those rules fall into the True Believer category and you need not

pay any attention.

The final clause in the rules for Followers concerns the files of certain projects which are worked on by several people on the same machine. We can't think of any of these right off the bat, but we have rules for them in any case. Analogously with users' home directories, there is the notion of a *project's* home directory, to be named ///Projects/*The-Project's-Name*/. For example, if there were several people doing coordinated work on Walnut on the same machine (presumably at different times), that work should take place in the subtree headed by ///Projects/Walnut/. As with users' home directories, only True Believers have rules for their contents.

That concludes that part of the True Religion which must be obeyed to be considered a Follower. We end this section on the local directory structure with a recommendation for the organization of home directory subtrees. We believe that this set-up makes life easiest but, in the final analysis, we don't really care whether you follow it or not, so long as you adhere to the mandatory precepts above.

The first subdirectory you might want to make in your subtree is a private Commands directory, analogous to the global one, ///®/Commands/. This is where you would put any specialized personal commands or modified versions of standard ones. The general rule on whether to put it in ///®/Commands/ or your personal commands directory is this:

> Thou shalt only place things in ///®/Commands/ which are the most recent versions of programs found in the <Cedar®> or <CedarChest®> directory for the current release, ®. You might also put there the most recent versions of programs found in the <DATools®> or other major public project directories --- consult the Primate of the domain involved.

This pretty much ensures that other people won't get very confused by what they find in ///®/Commands/. Remember, if you do set up your own commands directory to place it on the search path used by the Command Tool to look up commands. A discussion of how to do this was given above in the previous subsection.

The other kind of subdirectory we recommend you use in your home directory is for distinct tasks. For example, suppose one of the authors wanted to make some changes to the Chat program for logging in on remote machines. He would CD to a new directory, ///Users/Pavel.pa/Chat/, and do a BringOver of Chat.DF into that directory. All modification and testing of the program would take place there. Once it was all over with, he could then simply give the command "Delete *" in that directory and he would have cleaned everything up. Had he done this in a directory with other kinds of non-Chat files in it, cleaning up would have been considerably harder. Another advantage to using a separate directory for each programming task is that it provides you with a way to tell if your DF file is complete. Since your new subdirectory will presumably contain nothing but what you BroughtOver, if you can compile and run your program in that directory, the DF file is probably complete.

Thus end our recommendations concerning the organization of your private subtree and this section on the True Religion of Local Directories. Go forth and organize in peaceful coexistence!

My Configuration's OK, Your Configuration's OK

Back in the bad old days, the checkpoint was always not quite what you wanted. The Watch Tool was closed, but you want it open. Or maybe it was open, but you like it closed. The division between columns was at 600, but you (being of sound mind and body) want it at 512. The checkpoint had no "Compile" buttons in its Command Tools, but you find those buttons indispensible. The file system was also usually a bit askew. Walnut had been BroughtOver into

/// (with flagrant disregard to the True Religion outlined above), but you (a True member of the Church) want it in ///®/Commands/. Your favorite hacks were nowhere to be seen. Pay attention, we're talking about important stuff here!

In the good new days, the checkpoint will still be wrong, and file system may still be a little off. But upon RollBack, they will quickly (i.e., eventually) right themselves, *untouched by human hands!* In the following two sections, you will find: 1) how this works, 2) what you *must* do to be a good citizen.

How it Works, in Full and Gory Detail

This solution is just what you first think of: design sequences of commands that put your machine in your desired state, and arrange to have them executed at the right times. You could simply create command files, and explicitly invoke them at those times. What follows is a dark path that leads to the bright spot where you (almost) never have to explicitly invoke the configuring commands.

If you don't care so much about how and why this works, but only want to know how to take advantage of it, skim the following for examples, and tailor them to your tastes.

Getting ready to do useful work on a public Cedar machine involves many steps, possibly stretched out over months, where each step may generally be repeated many times after the preceding step. Here follows a description of the user-visible steps.

A New Release

It is still the unfortunate case that Cedar is distributed as one monolithic release. The creation of this release does not (directly) involve users. But its release definitely does! The first thing everybody must do is make themselves a new user profile, for use with that release. This is because users' profiles reference specific releases; the new profile, and every file reachable from it, must reference the new release. For more details, see the UserProfile documentation, and the Release Message of the release being updated to.

Some kind soul will install the new release on the public machines, in some inscrutable schedule that may occupy a stretch of time as vast as from a couple of weeks before the release to a few weeks after it. Such an installation often involves scraping the disk clean (figuratively speaking, usually), and installing shiny new files from the release into random places in the local directory structure. Sometimes the disk is not cleaned first, and you have to actually *think* about what to do.

In every release, there is a standard bootfile. A bootfile (together with some things like germ and microcode that are so arcane we won't even discuss them) specifies a very initial state of the virtual memory (can you say "virtual memory"? I knew you could). Doing a "full boot", the next step, involves using a bootfile.

A Full Boot and CheckPoint

A "full boot" starts by setting the machine to the state specified by the bootfile, and letting it go. It is currently in vogue to make bootfiles that immediately look for some more programs to start. A file called "Basic.Loadees" is searched for in some standard places, and the first one is taken. It lists some GNames ("global names" --- names of files on servers --- so you don't have to worry about what's on the disk so far, because there's almost nothing if this the first full boot of a new release) saying where to find these programs.

One of the Basic.Loadees is Our Friend, The Installer. The Installer interprets some user profile entries, so one of the first things the Installer does is to get clear exactly *which* profile it is to use. It asks something like "do you want to install a personal profile?". A "no" answer means it uses some default (which is usually sub-optimal for nearly every use, including public machines). A "yes" answer means it copies [User]<YourName>®>Top>YourName.Profile (actually, it gives you a chance to change its mind about this) to ///®/System/YourName.Profile (where "®" is the release number (e.g., "6.1"), and "YourName" is your account name), and makes that the profile that everything will use (until someone else tries to use the machine).

Some time later, the program that implements Command Tools perks up. It helpfully creates a Command Tool. It looks for a user profile entry named "CommandTool.BootCommands". This should contain a sequence of commands; they will be executed in that first Command Tool. These commands should put the file system and virtual memory in a standard vanilla state; the virtual memory state will be immortalized in the checkpoint and inflicted upon all subsequent users (until someone makes another checkpoint, hopefully getting it *right*). Here's an example:

```
CommandTool.BootCommands: "
    CD ///Commands/
    Bringover [Cedar]<CedarChest7.0>Top>Environment.DF
    ← ViewerOps.OpenIcon[icon: ViewerOps.FindViewer[\"Watch\"], bottom: FALSE]
    Clock
    DFTool
    Install BootTool
    "
```

The CD and Bringover are initializing the file system to the standard vanilla state. *If this user profile entry references the wrong version of Environment.DF, it will mess up the machine for everybody --- make sure you get this right.* Then some standard tools are run and the Watch Tool viewer is opened.

After all of these commands have been executed, everything should settle down. The machine is now ready for *un*orthodox use.

If you've actually been doing this, you will have noticed that a full boot takes a while (~2 - 10 minutes). When doing certain types of work in Cedar, your virtual memory gets terminally cluttered uncomfortably fast. A couple of years ago, They got tired of frequent booting. Thus were born Checkpoint and Rollback. Checkpoint saves a snapshot of the virtual memory. Rollback reinstates it. In each Cedar Volume (the world-swap debugger is one volume: the one you normally use is another), there can be no more than one checkpoint saved at any given time. It is polite to leave a vanilla checkpoint on a public machine when you are done: you may make an arbitrarily bizarre one for your own use, but you should replace it with a vanilla one before you leave.

A checkpoint can be made by issuing the "Checkpoint" command. This will initiate the process of saving a snapshot of the virtual memory. When that's done, it will be restored by a Rollback.

Rollback

A Rollback can happen because: 1) it's the last part of making a checkpoint, or 2) it's explicitly requested by command or button, or 3) the machine is booted, without the explicit request that it be a full boot.

This is one of the right times for a user's configuration commands to be executed. The

configuration commands can be split into two groups. One deals with configuring the CommandTool they run in, and these should be run in each CommandTool. The other group deals with everything else, and these should be run in only one CommandTool. Note that there can be interactions within a CommandTool between the commands of the two groups.

There are two sets of commands in the user profile attempting to do these very things. The first is called CommandTool.PerLogin and is run automatically in every existing Command Tool after every Rollback and every time you log in (i.e., come out of the "Idle" state). The second set is called CommandTool.NewUser. This set is run in exactly one Command Tool whenever you either Rollback or log in as a different user (i.e., the first time you start using the machine after someone else has been).

Here's an example CommandTool.NewUser entry:

```
CommandTool.NewUser: "
    CD ///Users/Spreitzer.pa/Commands/
    BringOver -p /User/Spreitzer/6.1/SpreitzerHomeCmds.df
    Promptery
    CD ///7.0/Commands/
    Install BootTool EditorComforts TiogaDWIM
    CD
    DeleteDFTools
    DFTool
    ← ViewerOps.SetOpenHeight[viewer: ViewerOps.FindViewer[\"EditTool\"], clientHeight:
            195]
    DoTiogaOps SaveSelectionA MatchCase MatchLiterally LeaveInitCap SubstituteInSel
            MatchAnywhere EqualLooksTest DoReplace TypeName \"code\" StyleName
            \"Cedar\" ReadProfile RestoreSelectionA
    "
```

First, these commands set up the disk according to this user's peculiar tastes by bringing over some non-standard stuff into a place that only that user should mess with. A non-standard tool is run from among the flotsam placed in that directory. This is followed by some commands that run some standard stuff. You'll note that some of it was run before from the CommandTool.BootCommands entry. Cedar will avoid running programs a second time (unless you explicitly ask to run it *again*), so this does no harm. This is helpful when this sequence is executed on a machine that was not initialized according to the earlier instructions. Finally, the EditTool and Tioga are seasoned to taste, using an interpreted procedure call and a special Tioga command. Note the invocation of "ReadProfile" in the "DoTiogaOps" command: in the current imperfect world, if this is not explicitly invoked, Tioga will continue to use the previous user's profile entries.

Strictly speaking, the above sequence should also have BroughtOver the standard stuff into the standard places. But that's an expensive operation, and not likely to be needed, so it can probably be omitted without endangering your immortal soul.

Here's an example CommandTool.PerLogin entry:

```
CommandTool.PerLogin: "
    ClearMenu
    CreateButton Open Open $FileNameSelection$
    CreateButton OpenR OpenR $FileNameSelection$
    CreateButton RLaTeX RTeX $FileNameSelection$
    CreateButton RTeX RTeX &Plain $FileNameSelection$
    AddSearchRule ///Users/Pavel.pa/Commands/
```

```
Alias Dir LS
Cookie
"
```

The first command here gets rid of any frivolous and silly Command Tool buttons that the previous user might have polluted the viewer with. The next 5 put up this users notion of the "correct" set of buttons. We then set up the Command Tool's search path to use the user's private commands, create an alias for the LS command (apparently, this user comes from a TOPS-20 environment), and fire up the fortune cookie apparatus.

After the rollback completes, the machine is ready for orthodox use.

Idling

Now that you've been through most of the fearsome task of "logging in", let's consider "logging out". Fortunately, it's rather easier. In the upper right corner of the screen is a button named "Idle". Hitting it is how you "log out". It turns the screen blank, with a dancing cursor that says "Type Key" (better attract sequences are coming soon, maybe). The next user (even if it's you again) must hit a key, and then supply name and password.

Making New CommandTools

Also in the upper right corner of the screen is a button named "Cmd". Hitting it creates a new CommandTool. One final set of commands, called CommandTool.EachCommandToolCommands, will be executed as the first thing that new CommandTool does. Usually, you will want to do the same set of things here as in your CommandTool.PerLogin entry. In order to allow you not to duplicate that stuff, though, there is a built-in command, "NotePerLogin" that will invoke that set of commands. Thus, in the usual case, your CommandTool.EachCommandToolCommands will be very simple:

```
CommandTool.EachCommandToolCommands: "
    NotePerLogin
    "
```

Just for completeness, there is also a "NoteNewUser" command, but nobody has found any good use for it.

What Everyone Must Do

Since Cedar is used for things other than public machines, not all of the above machinery is hard-wired in place. In order for everyone to share and enjoy swimming in the public pool, some care must be taken so as not to befoul other users.

If you don't make installations or checkpoints that others will be using, there's little for you to worry about. Just remember to not leave any booby-traps.

When making a checkpoint that will be left for others to use, you should include some standard tools. Some good examples are the Clock, the DF Tool, and the Boot Tool. The easiest way to get them running is through your CommandTool.BootCommands profile entry. When installing, you should make sure that the standard stuff is in the standard places. So far, that consists of BringingOver /Cedar/CedarChest®/Top/Environment.DF into ///®/Commands/. It is also easy to put that in your CommandTool.BootCommands. Finally, you should ***NOT*** put anything weird or not universally beloved into a checkpoint. Examples of such packages are FastMouse (Pavel, at least, finds it impossible to work with) and EditorComforts (there are still

some cretins in the world). The sample BootCommands entry given above is a shining example of how to do all of these things.

A Smug Concluding Section

In the revelatory sections above, we've outlined some truly elegant techniques for sharing a file system and checkpoint among users, and perhaps even given a few hints on how to organize tasks into directories (for more on this theological sub-field, see the Not-Very-Old Time Religion, in [Cedar]<Cedar6.1>Documentation>Cedar5LocalDirectories.Tioga, as written by the late St. Larry). Sharing among users may be less interesting for users of private machines, but sharing among tasks is not. Thus, we hope even the privileged many may find some small inspirational message herein.

The World being what it is (and just what the heck *is* it, anyway?), times and mores will likely change as we ignorant souls reach for enlightenment. If you think you've found some new hints of the True Religion, be you not quiet about it. Let the authors of this document know and mayhap we will see further editions for the greater edification of later generations of Public Cedar Machine users.