*Last Edited by: Jack Kent, May 8, 1987 1:01:00 pm PDT*

# Introduction to Cedar

## Version 7.0

**Abstract:** This memo is a sort of operators' manual for acquiring and using Cedar. It explains the minimum you need to to know about most things, depending upon other documents for the full story.

This memo is probably out of date if it is in hardcopy form. It is intended to document Release 7.0 of Cedar, March 1987, but some sections still reflect earlier releases.

**[If you are reading this document on-line, try using the Tioga Levels menu (if you can) to initially browse the top few levels of its structure before reading it straight through.]**

XEROX

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

## Introduction to Cedar: Contents

**0. Introduction**

**1. The Cedar World**

*1.0 Credentials*

*1.1 Screen Management and Input*

*1.2 Commander*

*1.3 Files*

*1.4 User Profile*

*1.5 Walnut*

*1.6 Maintain*

*1.7 Setting up your disk - a guide for the personal computer owner*

*1.8 Booting*

*1.9 General Failure Modes*

**2. Programming in Cedar**

*2.1 Running programs*

*2.2 Debugging*

*2.3 Locating software*

**3. References**

*3.1 General References*

*3.2 Cedar Language References*

## 0. Introduction

Cedar has a small, close-knit user community. Much useful information is not written down or appears in informal messages. To learn about using the system you must take some lessons from someone; get them to show you how to start a D-machine (Dandelions, and Dorados), how to use the mouse, etc. To work effectively you must keep in touch with what is going on. If you are using Cedar put your name on the CedarUsers↑.pa mailing list. (See the section 1.6 for instructions on how to do it.) If you are just generally interested in what is going on try CedarInterest↑.pa. Questions and bug reports should be sent to CedarSupport↑.pa. Questions about specific packages and problems should be addressed to the maintainers listed in the Catalog, with copies to CedarSupport.

In general, ask the people listed below when you have trouble with the system (names are listed in alphabetical order):

| | |
|---|---|
| Russ Atkinson | debugger, runtime system, general questions |
| Jules Bloomenthal | Viewers, Imager, Geometry3d |
| Alan Demers | communications |
| Tim Diebert | Tapes, archiving, Image scanning, Image processing, printer hardware |
| Ed Fiala | Cedar on DLions |
| Bob Hagmann | VM, Alpine, FS, File, SafeStorage, RPC, Summoner, general questions |
| Carl Hauser | Alpine, File, FS, general questions |
| Bill Jackson | Cedar on DLions |
| Christian Jacobi | ChipNDale, general design automation |
| Peter Kessler | runtime system, User Interface comforts |
| Willie-Sue Orr | Dorado microcode, Walnut, device heads, expert disk repair |
| Ken Pier | Remember, Icon Editor, Sil, Preview, Gargoyle |
| Michael Plass | Viewers, Tioga, Imager, printing, fonts |
| Mike Spreitzer | religious Guru, User Interface comforts, general design automation |
| Maureen Stone | printing, general Color in Cedar |
| Dan Swinehart | Etherphones, Finch, WalnutVoice, RPC, general questions |
| Doug Wyatt | Viewers, Tioga, Imager, CedarChest, general questions |

Typographical conventions employed herein:

Names of keys appear in capital letters in a small, alternate font: e.g., SHIFT, ESC, RETURN
↓ is sometimes used for RETURN.

Things that are typed or displayed on the screen appear in an alternate font; e.g., compile foo

When we describe fancy interactions in which a system completes commands, what you actually type is underlined: e.g., <u>Con</u>nect

Throughout this document, you will see references to "Cedar7.0" as part of a path name (e.g., "/Cedar/Cedar7.0/Top/BasicCedarDorado.boot"). If you are attempting to use this documentation for a later Cedar release, then replace the "7.0" with the release number.

## 1. The Cedar World

To use Cedar you must first find a Dorado or Dandelion.

Cedar will usually be found in its *idle* state, displaying the words "Type Key" in the cursor. After pressing a key on the keyboard, you will be asked to supply your name and password.

To awaken a slumbering ("powered off") Dorado press its boot button three times and wait for something to appear on the screen (a minute or so). The boot button is a little red button on the back of the keyboard between the mouse and keyboard cables.

To start Cedar on a wakened Dorado, boot the machine by pressing the boot button three times. Always triple boot Dorados when they are in an unknown state.

To turn on a Dandelion, power it up and wait for it to flash 206 (a few seconds). Do a "1" boot about a minute). For an "N boot" (N = 1 or 6 in the below instructions), press both buttons, and then releasing the "B RESET" button, while continuing to hold down the "ALT B" button. You will observe that the lighted digits begin to display a sequence of numbers: "0000", "0001", "0002", ..., "0010", "0000", ... repeating the sequence after "0010". For a "N" boot, release the "ALT B" button while a "N" is displayed. For an "N sub boot" (N = 20 in the below instructions), after a 6-boot, you will observe that eventually a "7777" is displayed in the lights. Once "7777" is displayed, again push the "ALT B" button. You will now observe that the following repeated sequence of numbers is displayed: "0003", "0006", "0020", "0021", "0022", ... "0027", "0003", .... For a "0020" sub boot, release the "ALT B" button while "0020" is displayed.

To start Cedar on an awakened Dandelion, do a "1" boot.

If Cedar is properly installed on the machine, then you will be prompted for a name and password or just a password; if you don't know what to type, get help. If the version number at the top of the screen is larger than 7.0, check for a newer version of this memo. If it is smaller, get the new release using the instructions in section 1.7.2. If the screen says Iago. . ., try typing Rollback and then RETURN when it prompts Cedar, or else Boot and then RETURN when it prompts Cedar, and enter f RETURN when it prompts Switches: . A Rollback is a lot faster than a Boot. Otherwise, you have no Cedar world: consult section 1.7 on how to get your disk set up. Get help if this is a public machine.

### 1.0 Credentials

Whenever you enter the Cedar world, you will be asked to supply your credentials: both Grapevine (Grapevine RName and password) and XNS (NS name and password). Providing the former is obligatory; Cedar expects its user to be an individual registered with Grapevine. Once you have been authenticated, Cedar will remember your credentials until you either (1) push the boot button, (2) boot a non-Cedar partition (e.g., an Alto partition), or (3) push the "Idle" button in the extreme upper right corner of the screen. If you are not registered with Grapevine, contact your local support staff.

For the nonce, most Cedar users will not (and need not) supply NS credentials. If you are such a person, just type two spaces (when asked for NS credentials), then answer no when asked if you want to try again. In the future, you'll find NS credentials will be quite valuable. (At that time, you'll be able to run Cedar with NS credentials alone and partake of two NS services (NSFiling and NSmailing) that demand NS credentials) To obtain them, users in the PARC domain should contact Carol <Lehner.pa>. Keep in mind that Login works most smoothly if you have an NS alias that corresponds to your Grapevine name (for example, Wyatt:PARC:Xerox corresponds to Wyatt.pa). In that case you can simply confirm the suggested NS name with a space. If your NS password matches your GV password, you can confirm that with a space too.

The precise form in which Cedar asks for your credentials depends upon the way in which the credentials were originally installed. Public machines and some personal machines (at their owners' option) have "unprotected" disks, meaning that Cedar will permit any individual recognized by Grapevine to log in. Other personal machines, however, have "protected" disks, meaning that Cedar will only allow a specific individual to log in. To change from a unprotected disk to a protected one, (or vice versa, if it is yours), use Iago's "Install Credentials" command.

## 1.1 Screen Management and Input

The basis for screen management is the *Viewer*. In general, a viewer manifests itself as a rectangular area on the screen. Some viewers simply display text, others are virtual *buttons* that invoke procedures when clicked. We use the verb *click* to describe the acts of positioning the mouse-controlled cursor over a viewer then depressing and releasing a mouse button, usually the left one. *Middle click* means to depress the middle button, *right click* means to depress the right button, etc. Buttons that are active are turned a stipple pattern (gray). In the past mouse buttons were imagined to be red, yellow, and blue (remember it by saying "row your boat") scanning from left to right, so you will occasionally see that terminology.

Across the top of the screen is a small *message area* where various comments about the system's status and behavior will appear. If the message is especially important, the message window will flash to call the your attention to it. The large middle part of the screen is divided into two columns for displaying tools and documents. Most tools and documents will initially appear as *icons*—small pictures at the bottom of the screen. You can *open* an icon to see its contents by middle clicking it; holding down the SHIFT key while clicking makes it consume the whole column. Left clicking an icon *selects* it, making it the recipient of type-in from the keyboard. Icon keyboard commands include:

| | |
|---|---|
| C | Move the icon to the color display (assuming you have the hardware). |
| DEL | Delete the icon. |
| L | Move the icon to the left column. |
| M | Move the icon to another column. |
| O | Open the icon (like middle clicking). |
| SHIFT-O | Open the icon full size (like SHIFT middle clicking). |
| R | Move the icon to the right column. |

Open viewers display a menu of commands across the top when you move the cursor into the caption (the black band at the top containing the name of the viewer). The commands are as follows:

| | |
|---|---|
| Destroy | Make the viewer disappear. |
| Adjust | Change the size of the viewer or size of the column (see below). |
| Top | Move the viewer to the top of the column. |
| <-- | Move the viewer to the left column. |
| --> | Move the viewer to the right column. |
| Grow | Close all other viewers in the column. |
| Close | Make the viewer iconic. |

Middle clicking in the caption menu always invokes the Grow command, and Right clicking always invokes Close. This allows you to invoke these frequently used operations without having to position the mouse as accurately.

The height of a viewer in a column is computed from a set of hints, some determined by programs and some indicated by the user. The program which created the viewer can specify a desired height (such as in the EditTool and Watch viewers) or the program can request that the viewer receive a "fair share" of the available space (as in Tioga text viewers). The user may override these program hints by clicking the *Adjust* caption menu command, entering a mode where a new height hint may be specified with the mouse. Moving the cursor out of the original column changes the mode to allow the user to specify a new column height and width. At any time while in adjust mode, simultaneously depressing two mouse buttons cancels the adjust command.

Some menu items and buttons will be displayed with a strikeout bar through the text. These are known as *guarded* commands, implying that command, if inadvertently triggered, might cause loss of your current state. To invoke a guarded command, click once to remove the guard and then again to trigger (within a few seconds or the guard will reappear).

In order to type something into a viewer, you must first establish the *input focus* by clicking somewhere inside it. (If the viewer is a typescript viewer, i.e., one in which you and the system alternately insert characters, as opposed to a Tioga document, you should make sure that you click the mouse in the white space below the last character.) Left clicking in a text area positions the blinking *caret* that indicates where your typed characters will appear. A sequence of text characters may be selected by left clicking the first character and right clicking the last: they will appear video reversed, i.e., white on black. Those characters then become the *current selection* which various buttons (e.g., Open) treat as an input parameter. Whatever you type replaces the current selection when it is video reversed. There are many other things to learn about selection described in the Tioga manual [G1]. Tioga is generally similar to Laurel [G2].

There are some buttons at the right end of the message area. You can boot any of your system volumes (e.g., Alto, Cedar, Debugger) by left clicking Boot to bring up a set of buttons. Right clicking Boot brings up a viewer that allows for more options in booting. The New button creates a new text viewer that you can type new text into: typing a file name followed by LF will load a file into it. The Open button creates a new text viewer for the file named by the current selection. Clicking Clean will take some older viewers and put them inside of a tool box labeled "MRU Cache". This is useful for getting icon space at the bottom of the screen. To get the viewer back, the "MRU Cache" icon may be opened, and the middle button clicked over the viewer's name. Clicking Idle causes the screen to turn black and display a dancing "Type Key" cursor. This is the preferred way of ending a session without actually leaving the Cedar world. If you subsequently hit a key, you will be asked to login, after which the screen will be restored to its state at the time that Idle was invoked.

It is a good idea to power off a Dorado at night since it takes almost 3 kilowatts to keep it running. Make sure the system is quiescent, then either do a "7-boot" (see section 1.7.4), or the PowerOff command or the PowerOff button.

The checkpoint facility allows you to save the effect of a long set-up computation such as the one that occurs after an installation of a release. Enter the Checkpoint command and wait for a while (over a minute on a Dandelion). After a bit, the screen will clear and the "simple terminal" window will appear. The system will type on this window to keep you appraised as to the progress of the checkpoint. Restoring the state saved in a checkpoint is called a rollback. Whenever you start your Cedar world or click the Rollback button, you will find yourself in the state saved in the checkpoint. It is important to understand that rolling back only restores the state of the virtual memory. It does not undo changes made to files or the directory. Thus you should create checkpoints only when the system is in a quiescent state with no open files: otherwise, strange things might happen after a rollback. If you want to overwrite a bcd (either by compiling or file transfer) that was loaded prior to a checkpoint you should boot the Cedar logical volume and create a new checkpoint.

The keyboard has a few unlabeled keys on a Dorado, and some peculiarly labeled keys on a Dandelion. Here is a list of Dorado keys, followed by the label on the Dandelion key which does the same thing:

| **Dorado** | **Dandelion** |
|---|---|
| LF | COPY |
| DEL | DELETE |
| BS | ← *(at top right)* |
| (LOOK) *(blank, second row)* | UNDO |
| (NEXT) *(blank, third row)* | KEYBOARD |
| (SWAT) *(blank, fourth row)* | STOP |
| ESC | CENTER |
| TAB | ⇒*(just above the LOCK key)* |
| CTRL | OPEN |
| ← | ' *(the right one)* |
| ' | ' *(the left one)* |
| ↑ | " *(the right one)* |
| " | " *(the left one)* |
| ~ | ¢ *(the cent sign)* |
| \ | DEFAULTS |
| \| | shift-DEFAULTS. |

## 1.2 Commander

Note: The following is summarized and extracted from the documentation of CommandToolDoc.tioga, [G15], which also appears as as a separate section of the Cedar Manual. For more complete discussion, refer to this documentation.

*General Comments*

The Cedar 7 CommandTool (also called the Commander) is a stream-oriented "glass teletype" command-line processor. It is much like the Unix (tm) shell, or the Alto executive, or the Tajo Executive, etc. Basically it is a place to invoke Cedar subsystems.

A new Cedar 7 world includes a gray screen with a few icons at the bottom and a few buttons in the upper right corner of the display. One of those buttons is labelled "Cmd". You can create a new top-level CommandTool at any time by clicking the Cmd button with any mouse button.

Whenever you boot Cedar 7, a CommandTool is automatically created.

The CommandTool viewer appears with the herald "Commander: WD = ///" The string after the equals sign is the current working directory of that command tool, much more will be said about this later and in the Commander documentation. The CommandTool will have printed a prompt (initially "% ") and will be waiting for input.

The menu of a CommandTool initially has three buttons, STOP!, Find, and Split. Find, and Split work the same way that the other Find, and Split buttons in the viewers package work. The STOP! button attempts to stop whatever is the current command running in the CommandTool viewer. It does this by raising the signal ABORTED in the CommandTool process. Usually this will result in the string " ...Aborted" being printed along with a new CommandTool prompt, but the details depend on the particular command, as well as other factors. Someday this will all be consistent.

There is a facility for adding more buttons of your own.

Documentation for individual commands is at present contained in files on the directory /Cedar/Cedar7.0/Documentation/. Each command has its own file; the command Mumble typically has a documentation file named MumbleDoc.tioga. Be leery of documents that do not have recent edit dates.

Detailed documentation for the CommandTool itself is located in /Cedar/Cedar7.0/Documentation/CommandToolStructureDoc.tioga. It includes a lot of information about the implementation of the CommandTool. As such, it will probably not be interesting to users, but instead will be a guide for those interested in implementing their own commands or in customizing the CommandTool itself.

*Facilities overview*

In general, the CommandTool interprets each line of input as a separate command. The first thing on the command line is taken to be the name of a command previously registered with Cedar. The rest of the command line is taken to be arguments to that command or directives to the CommandTool.

The previous paragraph is not entirely true, as it is possible to type several commands on the same command line and it is possible for the first thing on the command line to be the name of a command file, rather than itself be a command.

The rest of this section contains a brief list of the facilities provided by the CommandTool. Each facility will be described in more detail in the CommandTool documentation.

The CommandTool provides for:

*Initialization of itself*

The first instance of the CommandTool executes commands from the user profile entry (see section 1.4) CommandTool.BootCommands before becoming interactive. This will be done only when Cedar is Booted, not during a rollback.

```
    CommandTool.BootCommands: "Date\n"
```

All instances of the CommandTool which own viewers (you can see them on the screen, and this does not include the first one) execute commands from the user profile entry CommandTool.EachCommandToolCommands. This will be done whenever the "Cmd" button is clicked.

```
    CommandTool.EachCommandToolCommands: "
        AddSearchRule /// ///Commands/
        Statistics
        CreateButton SetBreak SetBreak $SelectedViewerName$ $ViewerPosition$
        CreateButton Compile Compile $FileNameSelection$
        CreateButton Open open $FileNameSelection$
        CreateButton Openr openr $FileNameSelection$
        "
```

See the individual command documentation to decipher this. (This is a good value for this entry to start with in your profile.)

When the user changes due to rollback or coming out of idle, the "NewUser" profile entry is interpreted.

"PerLogin" specifies command(s) that will be performed at each every change of user, every rollback, or every return from idle, and is done in every command tool.

Look in [Cedar]<Cedar7.0>Documentation> CommandToolDoc.tioga [G15] and HowToUseAPublicCedarMachine.tioga [G6] for a more complete description.

*@files*

A command line may contain "@filename". The CommandTool expands the given file in place as a macro expansion. @ files are different than command files, but often they will be equivalent. @ files cannot have arguments. @ files do not have to include any carriage returns, so they can be used as part of a single command.

*Command Files*

Command files are executed by creating a new, nested, instance of the CommandTool which reads from the given file rather than from the keyboard. Generally command files are files names Mumble.cm and they may be executed by just typing "Mumble" on the command line or by typing "Source Mumble", or "CommandTool Mumble." Command files can have arguments, see CommandToolDoc.tioga.

*IO redirection and pipes*

Commands are given a standard input stream and a standard output stream for their use. Normally, these streams are attached to the CommandTool viewer, but they may be redirected to read from or write to files. For example, the command line Date > date.txt will put the date and time into the file date.txt.

It is possible to "pipe" the output of one command into the input of another by use of the syntax commandA | commandB. For example, the command line Date | Indent 10 will print the date and time in the CommandTool viewer indented by 10 spaces.

*Star expansion*

A command line may include file system pattern matches such as *.mesa. The CommandTool provides facilities for expanding these patterns. At present, each CommandProc (which are the things that implement the commands) must decide for itself to expand *'s and call CommandTool.StarExpansion.

*Related commands*

There can be multiple commands on the same command line, separated by ';. If one of the earlier commands "Fails", later commands are not executed.

*Command Environment*

The CommandTool makes an attempt to set up interesting execution environments for commands. This area is still in flux. One of the most interesting properties the CommandTool can set is the Working Directory.

The CommandTool provides a STOP! button, which can halt many commands in their tracks. The effectiveness of this facility should improve. Some commands have substantial delay before noticing STOP!.

*Command lookup*

The CommandTool has a fairly elaborate and formal procedure for finding a CommandProc to execute in response to a user's command line. The procedures can be changed, but the default collection:

Uses search rules to find a CommandProc already present in the Commander registry. Search rules are a property of the CommandTool instance. They can be modified by the AddSearchRules and SetSearchRules, and printed by PrintSearchRules.

Automatically sets up programs which are not yet loaded (via .load files).

If a program, say foo, is not yet loaded, then the CommandTool looks for a file named foo.load. If it finds one, it runs it as a command file. When the command file is done, the CommandTool again uses the search rules to find a CommandProc now present in the Commander registry.

The CommandTool also automatically locates and invokes command files.

**Some Common Commands**

The following are some of the more useful standard commands. More commands can be discovered through the use of ? or ??. Other commands can be added by running the appropriate binary (.bcd) files. (The first three are pseudo-commands).

| | |
|---|---|
| @ | takes a file name as argument. Treat the contents of the named file as a command file, i.e. interpret the text as a sequence of commands. If {file} has no extension, look for a file of the form {file}.commands or {file}.cm. |
| ← | Treat the remainder of the input line as a mesa expression to be evaluated. Evaluate the expression and print its value. If the expression is terminated with ?, print the type of its value, rather than the value. If the expression is terminated with !, print the value showing the referents of all REFs and POINTERs to a greater depth. Note: this is particularly valuable to find implementations of procedures through interfaces (e.g., to find IO.PutF, enter "← IO.PutF": this yields "IOPrintImpl.PutF": now use open or openr to see the source for IOPrintImpl). |
| -- | Comment. |
| ? | Type a short description of all the commands that match this pattern. |
| ?? | Like "?" but includes commands that have not been loaded yet |

| | |
|---|---|
| AddDebugSearchRules | Add a new rule to the debugger search rules. |
| AddSearchRules | Add a new rule to the commander search rules. |
| Alias | Create an alias for a command. |
| Bind | Bind a list of configurations. |
| Bringover | Command to bring over some files using DF files (see section 1.3.3). |
| CacheKillExcessVersions | Deletes excess versions of files stored in the FS cache. This is a housekeeping program, and it helps your disk |

| | |
|---|---|
| | from getting cluttered with old versions of files. |
| CD | Change working directory. |
| Chat | Pup User Telnet, see ChatDoc.tioga (also section 1.3.4). |
| Checkpoint | Create a checkpoint. |
| Compile | Compile a list of modules. |
| CreateButton | Create a CommandTool herald button. |
| Copy | Copy newFile ← oldFile, or Copy directory ← list-of-patterns. |
| Date | Type today's date and time. |
| Debug | Tool for debugging processes and other worlds. |
| Delete | Delete a list-of-patterns. |
| Desktop | Creates a new desktop. |
| DFTool | User interface to DF files (see section 1.3.3). |
| FindR | Finds Cedar release source file names given the short names (.mesa extension is the default). |
| FindRBin | Finds Cedar release binary file names given the short names (.bcd extension is the default). |
| FlushCache | Flush FS cache. |
| GetFromRelease | Makes attachments for interfrace bcd files needed by the compiler and suggests changes to df files, all based on the error log from the compiler. |
| Help | Same as ?. |
| Interpreter | Create a new interpreter tool. |
| KillExcessVersions | Deletes excess versions of files. |
| List | Print names and versions of files matching the pattern (Say "? list" to get documentation about the switches). |
| ListBTree | List FS local directory and cache (the "FS BTree"). |
| Load | Load one or more .bcds – a Run without the Start. |
| Login | Login a new user. |
| Maintain | Performs inquiries and updates to the Grapevine database. |
| MakeDo | Automagically re-make a package. |
| Open | Open a viewer on the given filename. Also a button by this name. |
| Openr | Opens viewers on Cedar release source files and other files indexed by "version maps" (see section 2.3) given the short names (.mesa extension is the default). If a short name has multiple long names associated with it, the alternatives are listed, and no viewer is opened for that name. |
| PowerOff | Perform a somewhat orderly system shutdown, and then powers the machine off (Dorado's only). |
| Print | Prints a file (on paper!). |
| PrintDebugSearchRules | Print debugger search rules. |

| | |
|---|---|
| PrintSearchRules | Print commander search rules. |
| PseudoServerAdd | Add pseudo-server translation – note: changes not permanently recorded: they disappear whenever you rollback or boot. |
| PseudoServerPrint | Print pseudo-server translations. |
| PWD | Print working directory. |
| RemoveButton | Remove a CommandTool herald button. |
| Rename | Rename newFile ← oldFile, or Rename directory ← list-of-patterns. |
| Run | Load and Start the named programs. |
| SetBreak | Set a breakpoint in a specified source file and character position. |
| SetDebugSearchRules | Set the debugger search rules. |
| SetFreeboard | Set FS cache minimum free pages before flushing. |
| SetKeep | Sets the keep on a file. |
| SetSearchRules | Set commander search rules. |
| SModel | Command to save files using DF files (see section 1.3.3). |
| Start | Start a previously loaded bcd. |
| Statistics | Turn statistics printing on or off. |
| Unregister | Unregister a command. |
| User | Type the name of the logged-in user. |
| Walnut | For sending or retrieving mail. |
| Waterlily | Compare two source files. |

One command that probably does *not* do what you expect is Run. Run really means load and start a bcd file. That is, take the file output from the compiler or binder, load it into virtual memory, linkage edit all the externals, and start the module(s) inside the bcd. Modules may be optionally started inside configurations. The code that is executed is the body of the module(s). It may or it may not actually do anything. Normally, all that is done is that, after some initialization, a command is registered with the commander. Entering this command will then actually do the function. Thus, "Run Compiler" does not compile anything, it only loads the compiler into virtual memory (if it is not already there), and registers the "Compile" command.

For more details and a bigger list, see the "Command Index" section in
[Cedar]<Cedar7.0>Documentation>CedarCatalog.tioga and
[Cedar]<CedarChest7.0>Documentation>CedarChestCatalog.tioga

**Interpreter**

When typing to an Interpreter, the user is always prompted with &nn ←, where nn is the event number. What the user types following the ← is treated as an expression to be evaluated in the current context. The value of the expression will be assigned to the variable whose name precedes the ←, i.e. &nn. This value can be referenced in later expressions. As mentioned earlier, if ? is typed following an expression, the type of the expression, plus other explanatory information, is printed.

The following is taken from an actual session with the Interpreter. The italicized text at the right is added commentary not printed by Interpreter.

| | |
|---|---|
| **&2** ← Rope.Cat["Ce", "dar"]↵ | *Note, that Rope is the interface, not the implementation.* |
| => "Cedar" | |
| **&3** ← LIST["foo", "bar", &]↵ | *& evaluates to the previous result (&2 in this case.)* |
| => LIST["foo", "bar", "Cedar"] | |
| **&4** ← &3?↵ | *What type of list did the interpreter produce?* |
| => (type) LIST OF REF ANY | |
| **&5** ← &3.first?↵ | *What type is the first element of the list?* |
| => (type) REF ANY | |
| **&6** ← IO.PutF↵ | *Where is the implementation of the* PutF *procedure in the* IO *interface?* |
| => IOPrintImpl.PutF | |
| **&7** ← List.Reverse[&3]↵ | *Call the* Reverse *procedure in the* List *interface* |
| => LIST["Cedar", "bar", "foo"] | |

For more detailed information about exactly what subset of Cedar language expressions the interpreter can handle see [Cedar]<Cedar7.0>Documentation>InterpreterToolDoc.tioga.

## Event Viewers

Events occur when a program raises a signal or error that is not caught or encounters a breakpoint. Whenever an event occurs, the corresponding process is stopped so that it can be examined, and control transfers to a different viewer called a Event Viewer. An Event Viewer is an Interpreter whose default context is the context of the action. The user can then walk the stack and evaluate expressions. The user can also choose to ignore the action for the time being and type some other command in a different Event Viewer, CommandTool, or Interpreter. If the user does not wish to pursue the cause of the action at all, the simplest way to make it "go away" is to click Abort.

Event Viewers have an extra row of buttons as compared with an Interpreter. Clicking Source finds the source for the program (using the "Debugger Search Rules" established by the SetDebugSearchRules command and the version map for the release). The Abort button raises the abort signal, and will often kill the execution of the process. Proceed continues from the breakpoint, signal or error. Clicking ShowFrame displays the arguments and variables in the current frame (a procedure frame in the call stack). WalkStack moves up or down the call stack: a left click moves towards the bottom (goes towards the parent procedures), a middle click goes to the top ( most junior child), and a right click moves towards the top.

## Incantations to the Interpreter (ignore this on first reading)

To get the type index of a variable "foo":

SafeStorage.GetReferentType[foo]

To print the type of type index "typeIndex":

```
PrintTV.PrintType[typeIndex, &H.tsOutStream]
```

To print the type of a variable "foo":

```
PrintTV.PrintType[SafeStorage.GetReferentType[foo], &H.tsOutStream]
```

## 1.3 Files

All of the material you are working on, including programs, is stored in *files*. Each different document you handle will be stored on its own file. The file system is somewhat complicated by the fact that it spans a network and developed in an evolutionary fashion.

Files can reside in various places. First of all, there are local files. The only copies of these files are on the local disk. Other files can be on file servers. We currently run two different types of file servers: IFS (Interim File Servers such as Ivy, Cyan, and Indigo), and Alpine (Luther and Ebbetts).

### *1.3.1 Local and Remote Files*

The file system running on the workstation is called FS. It provides a uniform way to view all the files in the internet.

A file on your local disk is identified by its name, which is a string of letters (upper and lower case can be used interchangeably), digits, and any of the punctuation characters +- . $. By convention, a simple file name has two parts, which are called the *main name* and the *extension*; they are separated by a period. For example, "Introduction.tioga" is a file name, with main name "Introduction" and extension "tioga". File names cannot include blanks, or any punctuation characters except the ones just mentioned.

It is important to name your files in some systematic way, using the main name to identify it, and the extension to tell what kind of file it is. Unless there is a good reason to do otherwise, it is best to use one of the standard extensions given below.

Here is a list of extensions commonly encountered:

| | |
|---|---|
| .bcd | Cedar object program |
| .config | system configuration, input to binder |
| .cm | command file for the Commander or other programs |
| .doc | Tioga document (old convention) |
| .df | list of dated files for use in moving files between machines |
| .form | a form to be filled in for letters or mail |
| .interpress | an Interpress master, suitable for printing on an Interpress printer |
| .ip | a lazy version of .interpress |
| .mesa | Cedar or Mesa source code |
| .press | Press-format file, suitable for printing on a Spruce printer |
| .tioga | Tioga document |

The system doesn't care whether you capitalize letters in file names or not (i.e., ALPHA, alpha, and aLpHa refer to the same file), but it is a good idea to use capitalization to make names more readable. This is especially useful when a name consists of more than one word, since blanks are not allowed in file names: e.g., TripReport or MasterList.

*File servers* are large repositories for files. A file server's disk typically has many times the capacity of your local disk. Besides providing back-up for your local disk, they are the only reasonable places to put files you wish others to see or want to access yourself from different machines. The only reasonable way to do business is to keep your personal files backed up on a remote server. You should not rest easily unless the latest versions of all your important files are on a remote server somewhere.

In general (not on gateways, however), the name of file in network has the form

[Server]<directory>subDirectories>name.extension!version or

/Server/directory/subDirectories/name.extension!version

In general, the "[]<>" form is more universally accepted as it is the older convention. The "///" form is used interchangeably in almost all of Cedar, but you may get into trouble with older systems (e.g., if you Chat to an IFS (see below), you better use the "[]<>" form). Avoid names that are only a single character long; some selection operations get confused by single character names.

This form is sometimes called the *full path name*, and is called a *GName* by FS. The server is the name of the machine. Indigo, Cyan, and Ivy are the local servers; the first two are instances of IFS—the interim file system. The directory is the name of a project or person. Each user who has an account on a file server has his own directory, named by his user name. Files within a directory may be organized into *sub-directories* (except on certain remote servers, e.g., Gateways, Grapevine servers, do not have a directory structure or versions). For example, the file named

<Jones>Memos>ActivityReport.tioga!3

belongs in directory Jones, sub-directory Memos. You can have as many sub-directories as you wish within your own directory. You can even have sub-directories within sub-directories, to as many levels as you wish, subject to an overall limit of 99 characters in each file name (FS allows 120 characters in a file name). Subdirectories are entirely a naming convention based upon the use of the character >; there are no special operations for dealing with subdirectories on file servers. The use of subdirectories on the workstation file system, FS, is somewhat different and are described in [G16]. The name and extension serve the same purposes as on the local machine. When you put a file onto a file server, if there is already a file with the same name, the new file is added, with a version number one bigger than the old one. When you reference a file without specifying a version you get the one with the largest version number. As you can see, it is almost never necessary for you to specify a version number explicitly. Each file in the system carries a *create time*: the time when the content of the file was created. This attribute serves as a server-independent version stamp.

You can name a group of files by using file name *patterns* containing the magic character * which stands for any string of characters. For example, the pattern *.memo stands for all the files which have the extension "memo", and the pattern *.foo* stands for all the files which have "foo" as the first three characters of the extension.

## *1.3.2 The Workstation File System FS*

FS is a file system for use on a Cedar workstation. It provides access both to remote file servers and to the local disk. Remote files accessible from FS must reside on a file server that supports the FTP protocol, in particular most files reside on IFS's. These file servers may be accessed from many Cedar instances on different workstations at the same time. Local files are accessed through an abstraction called the local server. The local server is the set of logical volumes on the local disk of a Cedar instance. One of these logical volumes may be designated the system volume. FS provides a directory for each volume, and a cache for remote files on the system volume. The FS documentation gives a more complete description of FS [G16].

FS allows for binding of the name space from the local disk to remote files. Files can be added as *attachments*: you can use the local attached name for the name of the file, but it is really just an alias for the real full path name (GName). I can thus attach this file, "[Cedar]<Cedar7.0>Documentation>Introduction.tioga", to a local file named "Introduction.tioga" (in some subdirectory). This can be done by the copy command like this:

```
copy Introduction.tioga ← [Cedar]<Cedar7.0>Documentation>Introduction.tioga
```

This does not transfer the contents of the remote file to the local disk. It only establishes the name binding for Introduction.tioga on my local disk. Most attachments are done by the DF software that is described in the next section.

Now if I open "Introduction.tioga", FS sees if it has a cached copy of "[Cedar]<Cedar7.0>Documentation>Introduction.tioga" on the local disk. If so, it uses it. If not, it copies the file from the IFS server for "Cedar" (most likely Cyan) using the FTP protocol. The copy is put into the file cache, and the file is then opened. The file cache acts like a normal cache: it has cache validation and flushing algorithms.

Whenever a new file is written, it is *always* written to the local file system. To have the file also stored on a file server, it must be explicitly copied. Files can be copied by the "Copy" command (e.g.,

```
copy [Cedar]<Cedar7.0>Documentation>Introduction.tioga ← Introduction.tioga
```

), through a program interface to FS, and by the DF software (see the next section).

So in summary, are really three types of files: local, attached and remote. Local files reside only on your local disk. Attached files have local names for remote files. Attached files may or may not have entries in the file cache on the local disk. Finally, there are remote files. These are files named by their full path names. Remote files may also be in the file cache.

A variety of things can go wrong with the local file system. We can attempt to recover the "file name table" (we call it the "FS BTree") using the Scavenge command in Iago. If you get an error like "BTree locked in update," or if you get disk errors, then consult an expert. We have a variety of tools that we can use to repair disks. However, you should always "keep your bags packed" since we cannot recover from all errors.

### *1.3.3 DF Files*

As soon as you find yourself dealing with multiple files, you should devise *DF files* to help you back up and retrieve them. You should start using DF files very soon after you start using Cedar - you will be glad you did later. A DF file is a human-readable file that describes a list of files with their create dates. The simplest way to create a DF file is to list all the files of interest in a file and apply the command SModel to it. For example, to create the DF file describing all the files mentioned in a previous version of this memo we created the file init.df with the following content (updated to current naming conventions):

```
Directory [Cedar]<Cedar7.0>Top>init>
 Init.df
 AltoSupport.cm
 Cedar.cm
 D0.cm
 D0Release2.5.1.cm
 Dorado.cm
 DoradoRelease2.5.1.cm
 RunPilot.bcd
Directory [Cedar]<Cedar7.0>documentation>
 GettingStartedInCedar.memo
 GettingStartedInCedar.press
```

We then typed

SModel init

to the Commander. This is the same thing as using the DFTool with the standard options (access: all. origin: all. reference: all. selected files: (no filtering). action: enter). Now click (which selects) "SModel". enter "[Cedar]<Cedar7.0>Top>Init" in the "DF file(s)" line. clicking "Do It". and confirming the stores. This transferred all the files to the remote directories, including an updated version of init.df that contained the version numbers and create times of the files:

```
Directory [Cedar]<Cedar7.0>Top>init>
 Init.df
 AltoSupport.cm!1   22-Mar-82  9:47:24 PST
 Cedar.cm!1                 18-Mar-82 14:57:23 PST
 D0.cm!2                    22-Mar-82 10:38:37 PST
 D0Release2.5.1.cm!3        22-Mar-82 10:55:02 PST
 Dorado.cm!3                22-Mar-82 12:33:35 PST
 DoradoRelease2.5.1.cm!1    22-Mar-82 10:11:30 PST
 RunPilot.bcd!1     2-Feb-82 23:37:34 PST
Directory [Cedar]<Cedar7.0>documentation>
 GettingStartedInCedar.memo!1 22-Mar-82 14:02:23 PST
 GettingStartedInCedar.press!1 22-Mar-82 14:03:11 PST
```

Once you have a DF file, you can use the BringOver and SModel programs to manage the movement of your system.

BringOver init

makes sure the local disk contains the proper versions of the files in init.df by retrieving new versions automatically if needed. This is the same thing as using the DFTool with the standard options (Check existence on server: yes. Store changed files: yes). Now click (which selects) "BringOver". enter "[Cedar]<Cedar7.0>Top>Init" in the "DF file(s)" line. clicking "Do It". and confirming the entires. You should use full path names to specify a DF file if you want the remote version to control things: e.g.

BringOver [Cedar]<Cedar7.0>Top>init

After making changes to files, you can use SModel to write out the new versions: it compares the create times on the local files with those in the DF file and transfers files when they differ. You can nest DF files if your package requires another package. All of this is a simplification: see [G3] for more information. If you are providing a component of a Cedar release you must read [G3] to learn the proper way to use DF files for a Cedar release. You can learn a lot by looking at the DF files on [Cedar]<Cedar7.0>Top>.

This example was for use in the root directory, ///, of the local workstation. Public dorado gypsies should not use the root directory, but some subdirectory (e.g., "///Users/your-name.pa/").

A second way of doing "BringOver" and "SModel" is to use the DFTool. This is a tool that uses a button style interface to the screen.

The DF software is not all that good about parsing the last line of the file. Make sure the last line has a RETURN in it. It likes directory names in the "[]<>" format.

See section 2.3 *Locating software* to assist in building Include clauses for df files (see [G3] again.)

### *1.3.4 Chat and File Space Management*

Sooner or latter you will run out of disk space on your IFS directory. File space management activities not supported by the File Tool or DF files can be carried out by connecting to a file server with Chat. Chat uses a viewer to simulate a teletype computer terminal, and thereby enables you to talk directly to executive programs running in various server machines.

To initiate a conversation with the executive in a server type "Chat server-name" to the Commander (e.g., to chat to ivy, type "Chat ivy"). If all goes well, you will see a message from the server's executive and @ at the left margin prompting you for type-in. If Chat has trouble getting connected, it will tell you its problem after trying for a few seconds. This usually means that the server is broken or too busy; you might try again in a few minutes. To redirect an existing Chat viewer to a sever type the server name into a window, select the name, and click Login. If you click Connect rather than Login, you can login by hand: type

```
@Login (user) name (password) password
```

Whatever the server executive is doing, you can force it to stop by typing CTRL-C.

When you are finished talking to the server Executive, type

```
@Quit
```

If the file server is an IFS, you will be logged out automatically if you don't type anything for three minutes. This is because IFS can service only a small number of users (currently nine) at once; the automatic logout is intended to prevent IFS from being tied up by users who aren't doing anything useful. Simply closing a chat viewer does not shut down the connection unless you right click the Close button.

You may type ? at any point to obtain a brief explanation of what you are expected to type in next. An IFS displays the remainder of abbreviated commands.

To delete all old versions of files (i.e., all but the highest-numbered version of each file), on IFS type, after first logging on:

```
@Delete *,
@@Keep (# of versions) 1
@@Confirm (all deletes automatically)
@@
```

It is a good idea to do this fairly frequently, since old versions of files can pile up and waste a lot of space. Note that there is a "," in the Delete command above. To find out how much space you are using on the file server, type

```
@DskStat
```

One IFS page is equivalent to about four D-machine pages. You will notice that you also have a *disk limit* which is the maximum number of pages you are permitted to use on the file server at one time. If you exceed your disk limit, the server won't let you store any more files until you first delete some existing ones to get you below your disk limit. To get your limit changed, consult your local support staff.

If this doesn't make sense, then do the following. Type to the CommandTool "chat ivy". A chat window will appear, and the input focus (the upwards pointing blinking wedge) will be placed at the end. Type the above delete command and quit. The window will look something like the script below. What I typed is underlined, and RETURN is explicitly indicated. The IFS does command completion, so I really typed on the first line "d", "e" "l"," ", "*", ",", and "↲".

```
PARC Ivy IFS 1.38.1L, Executive of February 14, 1984; 3 users out of
9.
@Login (user) hagmann.pa (password)  (account)
@delete (files) *,↲
@@keep (# of versions) 1↲
@@confirm (all deletes automatically)
@@
<Hagmann>
  hagmann.df!338
  hagmann.df!339
  hagmann.df!340
  hagmann.df!341
  hagmann.df!342
@quit

Connection being closed by ivy.
```

You can direct your attention to some other directory by typing

@Connect (to directory) OtherDir (password) password

You may omit the password when connecting back to your own directory, or when connecting to a directory belonging to a project of which you are a member.

You can find more information about using Chat (mostly how the "List" command works) in the Alto User's Handbook, Alto Non-programmer's Guide, Section 6.6 and 6.7 [G13].

The Cedar Archive System (the *Archivist*) will save copies of files from file servers that use the Pup FTP Protocol (such as IFS's) onto tape. The copies can be later restored to the server. The interface to the Archivist is via electronic mail. An *Archive* or *Retrieve* request is encoded in a simple mail message, and mailed to "Archivist.pa". The message body will contain an entry of the following flavor for each file to be archived. An entry might be:

Archive: [ivy]<Diebert>Junk>*.*

Archive: [indigo]<Archivist>MessageLog.tioga!2

Archive: /ivy/Diebert/Junk/User.cm

In directory [Indigo]<Archivist>Forms>, see ArchiveRequest.form and RetrieveRequest.form for the forms to use in sending the mail. See [Cyan]<Archivist>Documentation>ArchiveDoc.tioga for more details on the Archivist [G19].

### *1.3.5 The Release Directory and CedarChest*

The release directory is [Cedar]<Cedar7.0>. Files within this directory are all part of the Cedar Release. Interesting subdirectories are Top (for .df files), Documentation, and the individual subdirectories for the packages (e.g., Compiler).

Much of the useful software in Cedar is not part of the formal release. The CedarChest7.0 directory provides a way for Cedar users to share software outside the formal Cedar release mechanism.

A entry is a useful (or semi-useful) program which is made available to the community of Cedar users as a public service. We call it the CedarChest because Dan Swinehart observed:

"When I was a kid the more valuable family possessions were stored in the Cedar chest. The Cedar lining was thought to have chemical properties that provided a protective environment (*environment*, get it?) for all these goodies. It's where the treasures were kept."

By creating CedarChest we (the Cedar administrators) hope to encourage Cedar users to share software; we also hope to reduce the current bulk of a Cedar release. Many packages currently inside Cedar belong more logically on top of it; over time, we plan to move these to CedarChest.

Though CedarChest is less tightly controlled than a Cedar release, it will work best if authors and users of the chest understand and follow some conventions. The Mesa developers have used a similar mechanism with success; many of our conventions are derived from theirs. Before using CedarChest, please read [Cedar]<CedarChest7.0>Documentation>CedarChestDoc.tioga.

### *1.3.6 Pseudo Servers*

Cedar depends on file servers for long term storage of common files. Since Cedar runs at more than one site, and since we would like not to be dependant on a single file server, we have *Pseudo Servers*. A Pseudo Server translation is a mapping from a logical file server name, to some real file server names. There is at most one write server (a write server of "$" means don't write), and a list of read servers.

Pseudo Servers may be set by the PseudoServerAdd (PSAdd) command and printed via the PseudoServerPrint (PSPrint) command. For example, to set up the Cedar server as writing to Cyan, but reading from Cyan or Luther.alpine, run the command:

```
PseudoServerAdd Cedar Cyan Cyan Luther.alpine
```

The Pseudo Servers currently in use in CSL are Cedar, Fonts and User. During installation, Pseudo Server "Cedar" is set to be as if "PseudoServerAdd Cedar Cyan Cyan" had been done and Pseudo Server "User" is set to be as if "PseudoServerAdd User Ivy Ivy" had been done. These can be listed and modified by the Iago commands (see below 1.7.2) "List Pseudo Servers," "Set Pseudo Server," and "Clear Pseudo Server". Changing the Pseudo Servers via Iago makes changes to a disk file that is read whenever the (logical volume) is booted (as opposed to when you rollback). Changes to Pseudo Servers via commands change the Pseudo Servers only for that boot or rollback of Cedar.

### *1.3.7 Local file system management*

Your local disk may start to fill up with files after you use system for a while. Or you may sit down at a public machine that has little free disk space (look at the "Free disk" display in Watch). Here are some hints of good commands to use.

CacheKillExcessVersions - CacheKillExcessVersions flushes excess versions of files in the file cache. Excess versions are all versions other than the highest numbered version of a file. CacheKillExcessVersions accepts a pattern (e.g., "CacheKillExcessVersions /Ivy/*" will kill excess versions from Ivy). By convention, it is always fine to use this command whenever you sit down at a public Dorado (Once, I flushed 200,000 pages off of a public machine using it).

KillExcessVersions - KillExcessVersions deletes excess versions of files that match the given patterns (* is the default). Excess versions are all versions other than the highest numbered version of a file.

ListBTree - ListBTree lists files matching a pattern in the FS cache. The first character of a line indicates the type of file (L means local, A means attached, C means cached). Local files print their names only, attached files print both LName and GName. Cached files print their GNames and the time last used.

FlushCache - FlushCache deletes files matching a pattern from the FS local disk cache.

PigsInSpace - PigsInSpace discovers the largest files in a logical volume. The normal system volume's name is "Cedar".

SetFreeBoard - Sets FS cache "minimum" freeboard. This is by default 1000 pages. The system tries to maintain free pages at least 90% of this freeboard. If the number of free pages falls below 90%, files are flushed in LRU order to move the free pages up to the limit. This is useful to get old files off of your disk: set the freeboard up, and when the dust clears set it back down to 1000.

ShowVAM - A program to show bitmaps in viewers of the volume allocation map (VAM) for any Cedar volume.

HistoVAM - A hack to analyze the distribution of free run sizes on a logical volume.

There are three other commands that are more for "wizards." Run FSUtil before issuing these commands.

LRUChain - Lists the current LRU chain for the FS cache of remote files. Each line contains the GName and the time last used. If you get a "LRU chain currently empty" message, flush one file by doing a "LRUFlush 1".

LRUFlush -

LRUFlush numberToFlush: Flushes the first numberToFlush files from the LRU chain. Files that currently are open are skipped.

LRUFlush -t hoursOld: Flushes all files whose used time is longer ago that "hoursOld" hours from the LRU chain of the FS cache of remote files. Files that currently are open are skipped.

LRUInfo - Lists statistics of operation for the FS cache of remote files.

## 1.4 User Profile

A number of components of Cedar permit the user to tailor Cedar's behavior along certain predefined dimensions via a mechanism called the *user profile*. Whenever you boot, rollback, startup a tool or certain other significant events, your user profile is consulted to obtain the value for (some of) these parameters. This operation is performed by consulting a file whose name is <YourName>.profile, e.g., Hagmann.Profile, or if no such file exists, User.Profile. This is the one file that can be kept in the root directory on a public machine. The entries in this profile are of the form

Key: Value

where, for any given key, the value is expected to be either TRUE/FALSE, a number, or a token (a sequence of characters delimited by SP, CR, TAB, COMMA, COLON, or SEMICOLON, or an arbitrary sequence of characters delimited by quotes), or a sequence of tokens. Comments can appear at any point in the profile, and are ignored.

More information and some examples may be found by examining [Cedar]<Cedar7.0>Documentation>UserProfileDoc.tioga, which lists all the currently available options. It is also enlightening to look at other people's profiles stored on the file servers.

## 1.5 Walnut

The program for reading and sending electronic mail in Cedar is called Walnut. It uses the database management system as a repository for the messages. The documentation for Walnut can be found in a file stored in the documentation directory [G18]; a copy of this documentation appears as a later chapter of this manual.

## 1.6 Maintain

Various administrative tasks associated with mail, authentication and other uses of Grapevine can be performed with the Maintain command. Bringover Maintain using the Bringover command or the DFTool (it is in Environment.df, and thus should be in your ///Commands/ directory so you don't really have to do a bringover). Its interface is layered according to the complexity of the operations various people need to perform. Many users will need only the level called "normal". This allows you to inspect distribution lists, add or remove yourself from lists, and change your password. When using Maintain, you must always specify names in full. Thus, you must say "CSL↑.pa", not "CSL↑", and "Hagmann.pa", not "Hagmann".

To look at a distribution list (a "group" in Grapevine terminology), fill in the text field labelled "Group" and click the "Members" button in the first line labelled "Type". The "Summary" button in that line will show you the access controls associated with that group (which control who may add or remove members). To add yourself to a group, fill in the "Group" field and click "Self" in the line labelled "Add". Similarly, you can remove yourself with the line labelled "Remove". Not all groups allow you to add or remove yourself. If you're not allowed to change the group, you should send a message to the owner of the group asking for the change. For example, you would send a message to "Owner-CSL↑.pa" to ask about "CSL↑.pa".

A summary of all the distribution lists is kept in Palo Alto on file [indigo]<Registrar>GV>FullDLMap.txt.

For the sake of security, it is a good idea to change your password occasionally (say, once a year). To do this, make sure your name is in the text field labelled "Individual", fill in your new password in the line below, labelled "Argument", then click "Password" in the line labelled "Set". Passwords should be at least six characters and unpronounceable. Almost everything is authenticated using Grapevine usernames (a Grapevine RName) and passwords.

## 1.7 Setting up your disk - a guide for the personal computer owner

This section describes how to obtain a Cedar world on your local disk. It is meant to be used only when installing Cedar on your personal computer. Do not do this on a public machine unless you know what you are doing.

If you have never installed Cedar before, you may want to get the assistance of someone who has. This is particularly recommended in the case of Dandelion users. We always recommend a full installation. Shortcuts are not recommended.

### *1.7.1 Cleaning up*

### 1.7.1.1 Save your private files on a server (if you have any files)

Before you proceed with the installation instructions given below, first clean up your old world and move the files you want to save to a file server. You can use "List -u ///*!h" in the CommandTool to list all files that have no known versions safely stored on a server. If the disk is of unknown state or scratch, ignore this step.

### 1.7.1.2 Make a new user profile

Cedar 7.0 uses a user profile entry to decide what DF files to bring over during booting. The boot looks in your user profile for an "Installation.Bringover" entry, which can be any list of DF Files; only the exported components of these DF files are fetched.

Whether you're upgrading from Cedar 7.1 or starting from scratch, it's helpful to store a 7.0 profile on your [User] file server before you install Cedar 7.0.

What if you're just getting started and don't even have a user profile on your local disk? In this case, the booting machinery will try to find a user profile in a reasonable place. First, if you have a directory on User, it will be searched for [User]<*name*>7.0>Top>*name*.profile. If this fails, then the default profile is taken from [Cedar]<Cedar7.0>Top>User.Profile.

If you already have a user profile for Cedar 7.1, edit it to make a new profile for Cedar 7.0; substituting "7.0" for "6.1" is a good start; also watch out for references to ///Commands/ (see "New directory conventions", above). If you don't have your own profile yet, use the file [Cedar]<Cedar7.0>Top>User.profile as a starting point. Store the new profile on [User]<*name*>7.0>Top>*name*.profile, where *name* is your user name. (Note the "Top>", which is new.)

If you plan substantial modification of the default profile, first look in

> [Cedar]<Cedar7.0>Documentation>UserProfileDoc.tioga

### *1.7.2 Installing Cedar*

Do be aware that *this procedure will erase your entire disk!* Before you proceed with the installation instructions given below, first clean up your old world and move the files you want to save to a file server. You can use "List -u ///*!h" in a CommandTool to list all files whose most recent versions are not safely stored on a server.

This is the recommended method for installing Cedar, even if you are already running in the Cedar 7.0 prerelease. Shortcuts are not recommended.

> ⇒ First, get to the CedarNetExec by using the standard NetExec to run the CedarNetExec. On a Dorado in an arbitrary state hold down BS, RETURN, and ' while triple booting. This places you in the Network Executive. The type-in conventions are simple: ? lists the possible commands, BS backspaces, DEL cancels the current line. To start up a program from the NetExec, simply type the name of that program followed by RETURN or ESC. Type "Cedar". After a net boot, you are in the CedarNetExec. Type "Cedar".
>
> Alternatively, for Dorados, triple-boot holding down the BS, RETURN and X keys. You do this by pressing the BS and RETURN keys, depressing the boot button (located between the wires on the back of the keyboard) three times, and then, in a timely manner, depressing the X key. Type "Cedar".
>
> If your disk has never been formatted, or is password protected and you are going to erase or reformat, use the Switches command to specify the N switch (for No disk) before typing

the last "Cedar". (You also can specify the N switch by holding down the "n" key between 812 and 845 during boot.)

The equivalent to the above on a DLion is to do a "0006 boot" (pronounced "six boot"), followed by a "0020 sub boot", where booting is explained in Section 1. A "0021 sub boot" is used when we are transitioning between incompatible releases. Cedar 7.1 and 7.0 are incompatible.

The above is called *Net Booting of Cedar* and it does not recover from errors very well. You may have to try again if it appears to get hung. It also has pauses (10-15 seconds) where nothing is displayed on the screen.

⇒ Log in. You must supply your Grapevine registered name (GV Name) and correct password, and your NS credentials before Iago (the "disk" utility) will permit you to do anything else. When asked, type your NS name and password. If you have no NS credentials, just type two spaces, then answer no when asked if you want to try again. If you don't have NS credentials yet, you should probably arrange to get them. (Users in the PARC domain can contact Carol <Lehner.pa>.) If the system won't let you enter your name and insists that you login as someone else, then you have a "protected disk". All you can do is to erase and rebuild the disk. A similar problem can occur on an unformatted brand new disk where there are disk errors trying to determine the disk's status. To overcome these problems, re-boot (the previous step) and using the "n" switch.

Login works most smoothly if you have an NS alias that corresponds to your Grapevine name (for example, Wyatt:PARC:Xerox corresponds to Wyatt.pa). In that case you can simply confirm the suggested NS name with a space. If your NS password matches your GV password, you can confirm that with a space too.

⇒ Alternatively, for those already running Cedar, get to Iago (boot with the L switch). Then install the Cedar 7.0 boot file via the Install Boot command, and specify that the boot file comes from one of (depending on the machine kind):

[Cyan]<Cedar7.0>Top>BasicCedarDorado.boot

[Cyan]<Cedar7.0>Top>BasicCedarDLion.boot

Then boot your machine with the L switch to get into a Cedar 7.0 world. You will be asked if you want to use Iago, answer *Yes*, and continue below skipping the "initialize your disk from scratch" discussion.

⇒ You will be asked if you want to initialize your disk from scratch: if your are in Palo Alto or are willing to use the default file servers in Palo Alto (incredibly slow over long distance), answer *Yes*. If you say *No*, you will be asked if you want to use Iago. This would allows you to do the installation yourself. If you have to change the pseudo servers to use (e. g., not in Palo Alto), answer *No* to initialize your disk from scratch. Always answer yes to using Iago. Ask someone locally (if you are not in Palo Alto) what good default remote names and pseudo servers are, then do the following:

> ▸ *Set up the default remote names.* You were told the systemHost, userHost and registry when Iago started. They should be "Cedar", "User" and whatever your Grapevine registry is ("pa" in Palo Alto). Use the "Set Remote Names "command to set these as needed. As each name appears, type in what you want. Hit return to leave it alone. These should be fine in Palo Alto. The only one that you should have to change is the registry if you are not in "pa" (e.g., "wbst" or "pasa").
>
> ▸ *Set up the pseudo servers.* You were told the pseudo servers known to the system when Iago started. Use the "List Pseudo Servers," "Set Pseudo Server," and "Clear Pseudo Server" commands to set these as needed. Use the Set Pseudo Server

command (once for each translation) to setup the following translations if you are in Palo Alto (do not be concerned with messages about inability to save the translations):

Cedar -r Cyan Cyan Luther.Alpine

User Ivy Ivy

Fonts -r Cyan Cyan Luther.Alpine

Summoner Indigo Indigo

DATools Cyan Cyan Luther.alpine

Do not worry if it says the translations couldn't be saved on the disk.

▶ *Continue with the installation.* Run the "Create User World" command. This continues below with the questions (How many Alto partitions?) as if you had answered yes above.

⇒ After saying yes, you will be asked questions to determine what operations must be performed to initialize your disk:

▶ How many Alto partitions? Usually you want 0 or 1 partitions (roughly 20K pages) for Alto emulation. This is the only way you can run Alto software on your machine. If you don't know what that is, then you don't need the partition. To have any Alto partitions, you must have at least one partition at high numbered addresses. If there is at least one high numbered partition, you may also have any number of partitions at low number addresses (up to the disk size).

▶ Is your disk is already formatted? Read the instructions carefully, but the most likely answer is yes. You don't want to reformat if you don't have to -- it's expensive. Owners of AMS-315 disks may want to reformat to try and detect more bad pages. The data pattern generator has been improved a lot for Cedar6.0 and later releases. We expect more bad pages to be found during format than was the case in Cedar5.2. First use the "Run Diagnostic BCD" command with the default argument ("/Cedar/Cedar7.0/Iago/ExtraIago.bcd") to get more commands. Then use the "List Bad Pages" command. **Write down the results.** Now format, and when you are done, list the bad pages again. To get the list to run properly, you will have to "Create Physical Volume" and "Create Logical Volume" to make a logical volume to span all of the Cedar formatted disk. "List Bad Pages" will then report the pages using both the logical and physical addresses. If any pages are missing from the original list, use the "Add (physical) Page to the BadPageTable" command to add the missing pages.

▶ Labeled or unlabeled? Unlabeled, probably. The implementor has the following advice for choosing to run labeled or unlabeled: if your machine can be tele-debugged from PARC, please use unlabeled FS (and keep your bags packed). If not, you are welcome to run it, but *really* keep your bags packed. [We expect that unlabeled operation will be the wave of the future, but we do not have enough experience with the file system to effect repair in case of error outside of PARC.]

▶ Password protect? If you say yes, only you can use the machine. Most users say no.

▶ VM size? Type RETURN to use the suggested size.

Note that clients who wish to have non-standard volumes will have to do the various steps by hand. See (or be) a wizard before attempting to perform a non-standard installation.

⇒ After you answer these questions, the list of the operations to be performed is printed and you will be asked for confirmation. If you made a mistake or changed your mind about how you would like the disk structured, don't worry -- nothing has been done and you can just repeat the sequence of questions and answers. Only after you give confirmation will the actions be performed.

⇒ Once the initialization has been confirmed, you can go get a cup of coffee (or two) while the volume(s) is/are initialized: no further confirmations are necessary. The installation will take roughly half an hour for a Dorado, somewhat more for other machines.

⇒ The boot sequence does allow for the Bringover of system and personal df files. Once booted, you should Bringover /Cedar/CedarChest7.0/Top/Environment.df into the ///7.0/Commands directory if it is not in your CommandTool.BootCommands entry in your user profile. You also want to Bringover all personal df files not in CommandTool.BootCommands that you really need for booting. Until all the Bringover's are done, you will have to use full path names for the binaries to be run (e.g., "Run /Cedar/Cedar7.0/DFTool/DFTool" to get the DFTool). If you got an error in booting that was caused by some name not being bound, do the Bringover's and re-boot. Re-booting can most easily be done by typing "run /Cedar/Cedar7.0/BootTool/BootTool", when it is done, left click the "Boot" button on the top of the screen, and then click "Cedar". In a timely manner, push down the "f" key. You can release the "f" key when typing appears on the screen. (Actually, the "keys" are set for booting are read somewhere during the transitions from MPCode 812 to MPCode 845. This is shown on the display panel of DLions, and in the cursor for a Dorado. The keyboard keys depressed during this time are set as boot keys. The "f" key means full boot: don't use the checkpoint even if you can find one.)

⇒ *When the installation finishes, boot your machine with the boot button (a physical boot).* This ensures that you are running the germ and microcode now installed on your disk. Do a *Full Boot* (see section 1.8).

⇒ Once your Cedar world has finished booting in a satisfactory manner, you should make a checkpoint (use the Checkpoint command).

⇒ Sometime later, you may want to re-make your checkpoint (e.g., since new versions of software are available). First, make sure that the proper df file(s) have been brought over or the df file(s) is part of the boot sequence by being in the CommandTool.BootCommands entry in your profile. Now *Full Boot* (see section 1.8), and after the dust settles (about two minutes on a Dorado) customize your Cedar system (if desired) and checkpoint it.

### *1.7.3 Other Iago commands*

Iago is the disk utility program for Cedar 7. It has a variety of commands for manipulating the local disk. To get to Iago, you can net boot it as above but answer *No* to initialize your disk from scratch, but say that you do want to use Iago. From a running Cedar world, you can boot to Iago by left clicking the "Boot" button on the top of the screen, and then clicking "Cedar". Hold down the "l" (lower case L) key after clicking "Cedar" until typing appears on the screen. You also can interpret "← IagoMainImpl.UseIago[]" to get to Iago from a running Cedar world. The Cedar world is still running, you are just now typing to Iago. This is for wizards only − you can really do yourself in using the more exotic Iago commands through this route. To get back the normal screen, enter the "quit" command, and confirm.

Typing "?" will display all the commands of Iago. Iago uses the abbreviated type-in mode: you only have to enter enough to make the command unique. Here are some common commands:

Boot Logical Volume: do a full boot of the installed boot file on the given logical volume

Check Drive: scan the drive and read all the pages: mark pages in error bad.

Copy File: copy a file: usually this is from/to an IFS to/from the local disk

Create Logical Volume: allocate some room from a physical volume to a new logical volume

Create Physical Volume: reinitalizes a physical volume to have no logical volumes

Create User World: this is the command that is done by default during the Cedar installation above. This does the following:

Formats if needed the disk.

Create a new physical volume

If there is a Debugger Volume:

- Create Logical Volume
- Erase Logical Volume
- Create VM Backing File
- Install Boot File

For the normal Cedar volume:

- Create Logical Volume
- Erase Logical Volume
- Create VM Backing File
- Install Cedar Microcode
- Install Cedar Germ
- Install Cedar Boot File
- Set Physical Microcode
- Set Physical Germ
- Set Physical Boot File
- Install Initial Microcode

Boot the normal Cedar volume

Create VM Backing File: sets the size and allocates the room for the backing file for virtual memory. The size of this file directly determines the size of your virtual memory. You can increase or decrease your virtual memory by using this command, and doing a full boot. Do not set the backing file size bigger than 16000 on DLions.

Erase Logical Volume: cleans all files off of the given volume

Format Disk: destructive write of all of the disk

Install Boot File: sets the given file as the file to boot when the "Boot Logical Volume" command is given. This will normally copy the file from the server.

Install Credentials: allows for password protection of the disk

Install Germ: the germ is a small bootstrap loader. This command installs one onto the disk

Install Initial Microcode: this installs the microcode to be used in initial physical booting

Install Cedar Microcode: this installs the microcode to be used in Cedar booting

Rollback Logical Volume: boot a logical volume by rolling back a checkpoint file

### *1.7.4 Common Iago Operations*

#### Extending Your VM Backing File

Use the Create VM Backing File command - see instructions above.

#### AMS-315 Disk Rebuilding

These disks have latent marginal bad spots. It is very important to run the formatter/verifier

(Format Disk) over them. This will destroy all the data on the disk, but is very useful in finding the bad spots. The default is 10 verify passes, but let it do 30 passes overnight. See the text under the bullet starting with "Is your disk is already formatted" above. After you have done all that it says, you will have all the bad pages in the bad page table. You can do the installation manually, but the easiest (but not fastest) thing to do is to use the "Create User World" command. This gets you to the identical point as net booting Cedar and answering "yes" to "initialize your disk from scratch". See the instructions above. Do not re-format during "Create User World" !!!

Recovering From Hard Disk Errors

See [Cedar]<CedarChest7.0>Documentation>DiskErrorRecovery.tioga

## 1.8 Booting

There are several types of boots associated with Cedar. There is a *physical boot* where the user actually pushes a physical button or moves some switches, and there is *soft booting* where the software boots the machine.

For a physical boot, the depressed keys at the time of the boot determine what gets booted. See [G5] for a complete description of Dorado booting. Except for when installing Cedar as in section 1.7.2 above, the Cedar user almost always boots without any keys depressed. This does the default boot, and for disks set up in the normal manner does a boot into Cedar.

When Cedar is booting, you can also set keys for the boot. This is done by depressing the keys when "812" is displayed, and releasing them when "845" appears. An "F" boot of Cedar is described below. Other common switches are "L" to boot to Iago, "W" that is sometimes used when rebooting the debugger volume (see below), and "N" for booting while ignoring the disk (an option for Iago). The case of the letter does not matter: don't push down the SHIFT key.

A soft boot is done under software control. This is normally done via the BootTool (the "Boot" button on the top of the screen), or Checkpoint/Rollback.

During a boot, several different types of files can be boot-loaded. An *etherboot* involves taking a copy of a boot file from a server and booting it over the ethernet. This is normally only done during Cedar installation. Etherboots often fail and should be retried after an appropriate interval. Booting a Cedar world has three major options (to the novice): *boot to Iago*, *Full Boot*, and *Rollback*. A boot to Iago is described above.

A *full boot* loads the *Basic Boot File*, and starts it up. To trigger a full boot, boot a Cedar volume where there is no checkpoint, or do the boot with the "f" switch. The switch can be set by: 1) holding down the "f" key during booting from the time "812" is displayed until the time when "845" is displayed, 2) using the BootTool, by right clicking the "Boot" button on the top of the screen, and setting the "f" switch, or 3) booting a Cedar volume from Iago (the Boot Logical Volume command) and specifying the "f" switch. The basic boot sequence then looks for a "Basic.loadees" (normally in /Cedar/cedar7.0/top/Basic.Loadees), and runs all the binaries found. The end of this sequence does a "Bringover" of BootEssentials, and a "Bringover" of all the df files specified in the user's Installation.Bringover entry in the profile. When all the dust settles, it is a good time to make a checkpoint.

Full boots are dependent on the system file server (Cyan in Palo Alto) being up. Any "FS.Error" occurring during full boot means that either a file server is down, or it was overloaded. Retrying may succeed if the server is only overloaded.

A rollback is the restart of a Cedar image stored on disk. The image was written by Checkpoint. Booting a Cedar volume, with no switches, where there is a Checkpoint file, will cause the Checkpoint to be booted. This is the normal method of booting and occurs when you boot from the physical boot switch or by clicking RollBack.

The debugger volume, if you have one and use it, is somewhat different (normal users do not need to read this paragraph). It also is built during Cedar installation. When a world-swap error occurs, the running Cedar world is *outloaded* (written to a special place on the disk), and the Debugger world is restarted (this is called *inloading*). This is similar to a RollBack except that when you leave the Debugger by an approved route (proceeding the Cedar world or booting something with the BootTool), then the current state of the Debugger world is saved as the image to restart the next time the Debugger is needed. The Debugger will sometimes get confused (some cached values are wrong?). You can reboot the Debugger with the "w" switch ("w" means to assume debuggee outload file is valid/interesting), and continue debugging. Of course, you loose the typescripts and everything volatile from the previous debugger. Leaving the Debugger by a non-approved route, such as a physical boot, will cause the debugger to be full booted the next time it is needed. The use of the world swap debugger has almost been discontinued due to the long time to swap between worlds, and the large amount of disk space needed to keep the worlds.

All of these variations are called booting. If someone tells you to boot something, ask for clarification if it is not clear what type of boot is meant.

## 1.9 General Failure Modes

If the system file server(s) is (are) down (Cyan, Luther.alpine, and Renegade.alpine in Palo Alto) then Cedar will exhibit peculiar behavior. Sometimes this is only annoying, such as opening some files is very slow, but often "server inaccessible" errors will prevent useful work. To see if Cyan is up, boot Iago and use the "List Files" command to list a file on the server such as /Cedar/Cedar7.0/Top/FS.df. If you are using a running Cedar world, type "Chat Cyan" to a CommandTool window. If you cannot open the connection to Cyan, then there is trouble. If you can open the connection, click "Disconnect" and destroy the window. You cannot full boot when all of the system file server(s) is (are) down, and you cannot do a Bringover of any release software.

If the user file server is down (Ivy in Palo Alto), then you cannot access your files or save changed files.

You also may have the misfortune to encounter a bug in the Cedar system that causes it to crash. There are various ways to recover from crashes.

If you seem to be stuck and the maintenance panel lights (on a Dorado they actually appear on the screen) say:

800: Empty MP: Cedar in normal operations. On a Dorado, this is signified by having the cursor something other than a number.

810: This is displayed while booting or world-swapping and does not indicate a failure.

811: Writing an outload file such as a checkpoint. This is a common code during worldswap debugging.

812: Reading an outload file (such as a checkpoint).

815: Cedar tried to transfer control to a world-swap debugger, but there isn't one on your local disk. See section 2.2. This can be forced by depressing CTRL-LOOK-SWAT on any machine, or just CTRL-SWAT or CTRL-LEFTSHIFT-SWAT as described below.

821-829: Various bootstrapping errors.

840: Initialization started -- not an error.

845: Mesa runtime initialization - this is the time during booting where the keyboard is

looked at to see the "boot switches" -- not an error
850: Virtual memory initialized during booting -- not an error
855: Storage initialized during booting -- not an error.
860: File initialized during booting -- not an error.
865: Communication initialized during booting -- not an error.
870: FS initialized during booting -- not an error.

Do an "openr MPCodes" for more codes. (If your machine is dead, you may need to do this on a different machine).

An 815 MP code indicates that the machine is waiting for teledebugging. Find a machine running (the same release of) Cedar, and ask if the user will teledebug your machine. It is a good idea for them to save their work, since the teledebugger has been known to crash. Get into the DebugTool by typing "Debug machine-name" where machine-name is replaced by the machine's name stuck in 815 (e.g., "Hornet"). If everything works, then an "Event" window showing an error from the machine in 815 will appear on the display. If you forced the machine into 815 (e.g., CTRL-LEFTSHIFT-SWAT), then this will be the DebugNub and not really the error process. Use the DebugTool, described in [Cedar]<Cedar7.0>Documentation>DebugToolDoc.tioga to poke around and find the error. Freezing "SignalsImpl" is a good way to find processes that have current signals or errors raised. Using the "SetDebugSearchRules" and "AddDebugSearchRules" commands may help the system to find source and binary necessary for debugging. See 2.2 and 2.3 below.

Anything less than 800 typically indicates a microcode or hardware problem. Maintenance panel codes less than 800 usually occur only on DLions. On a Dorado, a hardware or microcode-detected error usually halts the machine; the usual manifestation is that the screen turns completely black or gray with diagonal stripes or vertical lines. Seek help from the hardware support staff if this occurs (mail to CMS.pa or call 4370 at PARC): it probably means you have a bad RAM chip in the microcontrol store and they can find it if you don't reboot. Maintenance panel codes above 800 but not in the above list are usually but not always due to hardware problems also.

If the maintenance panel says 800 (or, on a Dorado, no numbers are visible), but there is no response to keyboard or mouse input, you need to get control somehow. In general, there are several levels of fall-back to try. Starting from the least drastic:

0. Click a Stop button or type CTRL-DEL in a viewer. This will sometimes stop a run-away program and get the system behind the viewer to listen.

1. Go to a world-swap debugger by typing CTRL-SWAT; the SWAT key is the unmarked one next to the right SHIFT key. You can look around and then resume without losing anything. If your machine doesn't have a world-swap debugger installed, this may produce a maintenance panel code of 815, as described above.

2. Type CTRL-LEFTSHIFT-SWAT to get to a world-swap debugger "delicately". As with the previous step, this can produce a maintenance panel code of 815. This is more violent in forcing you into the world-swap debugger, and you cannot call functions in the debugged world.

3. Perform a rollback (or boot if no checkpoint file exists) using either the Rollback button or the physical boot button. This loses whatever is in virtual memory and open files.

4a. On Dorados, boot by a three button boot. This fetches new microcode from the disk. Now perform step 3.

4b. On DLions, do a "1" boot.

If the above methods fail to get your Cedar World up, there are problems with your file system.

5. Flush the file cache. Do this by doing the Iago "Flush Cache" command on all files. This normally involves flushing "[Cedar]*" "[Ivy]*" and "[User]*".
6. There are various scavenging procedures available under Iago: Check Drive and Scavenge (although this might not work) . Get an expert to help you with these. These might recover the situation without much information loss.
7. Erase your client volume and get a new release, losing all non-backed up files. See section 1.7.3.
8. Initialize your disk and get a new release. See section 1.7.2.

## 2. Programming in Cedar

### 2.1 Running programs

You might try running the following example program:

```
-- Test.mesa
   last modified by Jim Morris July 8, 1982 12:31 pm
   last modified by Bob Hagmann April 30, 1984 2:45:41 pm PDT
   DIRECTORY
      IO USING [GetInt, PutF, int, STREAM],
      Commander USING [CommandProc, Register],
      ViewerIO USING [CreateViewerStreams]
      ;

   Test: CEDAR PROGRAM IMPORTS Commander, IO, ViewerIO = BEGIN
   in, out: IO.STREAM;

   Compute: Commander.CommandProc = BEGIN
      i, j: INT;
      out.PutF["Type me a couple of numbers: "];
      i ← in.GetInt[];
      j ← in.GetInt[];
      out.PutF["The sum of %g and %g is %g.\n", IO.int[i], IO.int[j], IO.int[i+j]];
      END;
   [in, out] ← ViewerIO.CreateViewerStreams["Compute.Log"];
   Commander.Register["Compute", Compute];
   END.
```

To create this file, take a CommandTool and "cd" it to a new directory. Enter the "New" command, and copy this text into the new viewer (see the Tioga documentation [G1]), and store the file as test.mesa (using the Files sub-menu: select the name to store it under (Test.mesa), left click "Store", and then click it again when the message window displays "Confirm Store to file ...").

To establish the file name binding for the interfaces needed in the compilation, use the commands listed belod. When you are all done, do a "delete *!*" in this directory (not in "///"

!!!!!!) to clean it out. Now type:

```
bringover -mp IO BasicPackages ViewerIO Rope
```

You then compile and run it with the following interaction:

```
% compile test
Compiling: Test/-g . . . . . . no errors
End of compilation
% Run test
Loaded and ran: []<>Test.bcd!1
% Compute
```

Now click anywhere near the bottom of the new viewer named Compute.Log, and type in two numbers, followed by a RETURN.

Compiler error messages can be somewhat obscure. Look in [Cedar]<Cedar7.0>Documentation>CompilerMessagesDoc.tioga for assistance in understanding them.

## 2.2 Debugging

DebugTool [G4] provides the Cedar debugging facilities, which include a basic interpreter, primitives for controlling programs by setting breakpoints, proceeding from breakpoints, and other such services. These facilities are available to the user through the Commander as described in section 1.2. The BreakTool [G14] provides more complicated facilities such as conditional breakpoints and conditional tracing.

### *Interpreter*

The interpreter has access to all names defined in the global frames of loaded programs (including types) plus all names in a special name space local to the interpreter. These special names all begin with &.

The interpreter handles a subset of Cedar expressions. The following is a summary of the subset:

| | |
|---|---|
| constants | fixed, REAL, Rope.ROPE, CHAR, BOOL, enumerated |
| simple variables | evaluated according to search rules below |
| x.y | x is a RECORD, REF or POINTER TO RECORD, global frame |
| x[y] | x is a SEQUENCE or ARRAY, REF or POINTER TO SEQUENCE or ARRAY |
| P[args] | P is a PROCEDURE taking given arguments |
| RT[args] | a RECORD constructor where RT is a RECORD type |
| LIST[exprs] | evaluates a list of expressions, producing a LIST OF REF ANY |
| X ← Y | X and Y are expressions |

The interpreter handles expressions containing arithmetic and logical operators (such as + and OR), and conditional expressions (IF but not SELECT.) Sometimes it is necessary to write parentheses around an expression to prevent the interpreter from getting confused. In general, you must prefix a procedure name with the name of its interface or implementation module: e.g., Rope.Cat.

In looking up a name, the interpreter

Checks to see if the name begins with &, and if so it binds to the named &-variable.

Otherwise, it searches the naming context context. Naming contexts may be a local frame, a global frame, an interface record or a world. These can be set by the &slf, &sgf, &sir, and &sw interpreter function calls.

*Search rules ("No Source..." and "No Symbols..." errors)*

To use the debugger and interpreter, the system must be able to find the .bcd and .mesa files for the parts of the system you are looking at. The "debugging search rules" specify a list of directories to look for software. Use the PrintDebugSearchRules, AddDebugSearchRules, and SetDebugSearchRules commands to see or modify the rules. When you run a .bcd, the directory of the .bcd is added to the debugging search rules.

If the system is looking for a file named *fileName* (e.g., foo.mesa or foo.bcd), then it will look in the directories in the debugging search rules in order and see if *fileName!h* matches the create date or version stamp it is looking for. The !h is *very* important: it will not "see" old versions. If it does not find *fileName* in any of the directories, then it consults the version maps. These are specified by the User Profile entry VersionMap.SourceMaps and VersionMap.SymbolsMaps. It examines these maps, and hopefully will find the files.

What can go wrong?

You can forget to run the .bcd. The debugger is only willing to work with loaded code.

You can edit and/or re-compile. This will mess up the version stuff.

You can re-run a program without clearing all the breakpoints. You can then hit a old breakpoint and not be able to see the source.

You can name the module something different than the file name. The name in front of the PROGRAM or MONITOR keyword suffixed with ".mesa" *must* (!!!) be the name of the file.

*CoCedar*

CoCedar is the world-swap debugger for Cedar. If you find yourself in a situation which seems to require knowledge of CoCedar, you probably should find a wizard.

## 2.3 Locating software

The most important document for locating parts of the system are the Catalogs ([Cedar]<Cedar7.0>Documentation>CedarCatalog.tioga [G9] and [Cedar]<CedarChest7.0>Documentation>CedarChestCatalog.tioga). These files list the various packages with brief descriptions and pointers to further documentation (if available).

If you are reading some software that is loaded in your world (most commonly by the run command or in the basic boot file), then use the "←" operator in a CommandTool to find the implementation. For example, suppose I have a source line like this:

IO.PutF[cmd.out,"Variance %g\n", IO.real[var]];

To find the implementation of PutF, enter "← IO.PutF" into a CommandTool window. This yields "IOPrintImpl.PutF". Now use open or openr to see the source for IOPrintImpl (e.g., type "openr IOPrintImpl", or select "IOPrintImpl" and click "openr" if you have a button for it). Use open for files know in the current subdirectory, and openr for files in the release (such as IOPrintImpl). Select "PutF", and click "Def". Alternatively, select "IOPrintImpl.PutF" and click "openr" (or type "openr IOPrintImpl.PutF") to do all this in one step.

Things are much harder if the software is not loaded in your system. If it is in the Cedar release, then it is indexed in [Cedar]<Cedar7.0>Documentation>DFIncludes.txt. Open this file and do a Find on the package name (e.g., "IO.bcd" for the example above). This will tell you the df file where the interface is exported. Normally, it is pretty straightforward to find or guess where the modules are that export to the interface (for interface Foo.mesa, look in FooImpl.mesa). If there is more than one candidate, you can either open them all, or use grep to search them (grep looks a lot like the UNIX (tm) grep). You may have to do a bringover on grep and run it before using it.

While doing compilations, you will often get errors due to missing interface BCDs (interface 'cannot be opened' in compiler jargon). GetFromRelease helps remove these compiler errors by making attachments, and suggests additions to the df file. GetFromRelease reads the Compiler.log (and hence any Foo.errlog files if you use separate logs) looking for the interface names that are missing. GetFromRelease uses the Cedar version maps to discover the released files containing the interfaces.

Findr, FindRBin and Openr use the *version maps.* This is how much of the system maps from short file names to global file names. There are maps for both source and symbols, and they are set by User Profile options VersionMap.SourceMaps and VersionMap.SymbolsMaps. A good starting value for these are as follows:

VersionMap.SourceMaps: ///CedarSource.VersionMap ///Commands/CedarChestSource.VersionMap

VersionMap.SymbolsMaps: ///CedarSymbols.VersionMap ///Commands/CedarChestSymbols.VersionMap

We have assumed that you have done a Bringover into the ///Commands directory of /Cedar/CedarChest7.0/Environment.df. Note that these directories for these version map files are established by conventions. The two files in your root directory get there from BootEssentials.df whenever you full boot, and the two files in ///Commands/ come from Environment.df.

Cedar.dfIncludes is a text file that shows where files in the release are included in the df files, and is useful for setting up the "Imports" section of a df file [G7]. Another file of interest in [Cedar]<Cedar7.0>Documentation> is Cedar.depends [G8]; it shows the compilation dependencies of files in the release. For CedarChest, look in [Cedar]<CedarChest7.0>Documentation>CedarChest.depends.

Things are much harder if you are trying to find some logical entities (e.g. all packages that provide some data structure). You can scan the Catalog, "List" inside of [Cedar]<Cedar7.0>Top> and [Cedar]<Cedar7.0>Documentation>, and the CedarChest7.0 directories, or ask someone.

To find the implementation of a registered command, such as openr, do the following in a CommandTool window (in the proper subdirectory):

```
% ? findr
///Commands/FindR    Finds Cedar release source file names given the short names
(.mesa extension is the default)
{implementor: ↑OpenRCommandsImpl.FindCommand}
```

Now do an "openr" on OpenRCommandsImpl.FindCommand.

## 3. References

In general, the following directories are worth browsing:

[Cedar]<Cedar7.0>Documentation>* is the general repository for documentation.

[Cedar]<Cedar7.0>Top>*.df will list pointers to things in the release.

[Cedar]<CedarChest7.0>Documentation>* is another repository for documentation.

[Cedar]<CedarChest7.0>Top>*.df will list pointers to things in the Cedar Chest.

There are subdirectories of these directories depending on the package or subsystem involved. Use * liberally when in doubt.

### 3.1 General References

In the following, assume the file is on [Cedar]<Cedar7.0>Documentation> or [Cedar]<CedarChest7.0>Documentation> unless a full path name is explicitly given

[G1] Paxton, W., *The Tioga Editor.* In the Cedar Manual and TiogaDoc.tioga. The manual for the text editor and manuscript preparation system.

[G2] Brotz, D., *Laurel Manual*, CSL-81-6. This is a CSL "Blue and White".

[G3] *The DFTool*, stored as DFToolDoc.tioga.

[G4] Russ Atkinson, *The Cedar Debug Tool*, stored as DebugToolDoc.tioga.

[G5] Ed Taft, *Dorado booting operation*, stored in DoradoBooting.tioga. A guide to the somewhat complex booting of a Dorado.

[G6] Pavel Curtis and Mike Spreitzer, *How To Use A Public Cedar Machine*, stored as [Cedar]<Cedar7.0>Documentation>HowToUseAPublicCedarMachine.tioga. 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.

[G7] Cedar.dfIncludes is a text file that shows where files in the release are included in the df files. This is useful for setting up the "Imports" section of a df file.

[G8] Cedar.depends and CedarChest.depends are files showing the compilation dependencies of files in the release. This is useful to determine the impact of some change on its clients in the release.

[G9] Horning, J. (ed.), *The Cedar Catalog*. In the Cedar Manual and CedarCatalog.tioga. A description of programs available for use and study. Look in [Cedar]<CedarChest7.0>Documentation>CedarChestCatalog.tioga too.

[G10] Doug Wyatt, *The Release Messages*, stored as ReleaseMessage.tioga (older ones in /Cedar/Cedar6.*/Documentation). This message describe the properties of each new release. They often contain vital pieces of information about known bugs and how to avoid them. Obviously, the information in older messages may be out of date.

[G11] Paul Rovner, *Cedar InterpreterTool*, stored as InterpreterToolDoc.tioga. The InterpreterTool is a typescript-style tool with a read/eval/print control loop at the top level. It is used primarily for debugging and testing, but it is also used for determining and occasionally changing status variables in a running system, perhaps one a world-swap or teledebug away.

[G12] Ramshaw and Larson, *The Briefing Blurb: Exploring the Ethernet with Mouse and Keyboard*, BriefingBlurb.tioga. Describes (almost) everything there was to know about the basic CSL computer environment in 1984.

[G13] Lampson, B., Taft, E., *Alto Users Handbook*, November, 1978. The basic reference for using the Alto system.

[G14] Russ Atkinson, *Celtics & BreakTool -- Two Cedar performance & debugging tools*, stored as [Cedar]<CedarChest7.0>Documentation>CelticsDoc.tioga.

[G15] L. Stewart, *Command Tool Guide*, stored in CommandToolDoc.tioga. This document is a user's guide for the CommandTool in Cedar 7.

[G16] M. Schroeder, *File Access from a Cedar Workstation*, stored in FSDoc.Tioga. This document describes the standard facilities available for file access from a Cedar 5 and 6 workstation.

[G17] Mark Brown, *The IO and Convert interfaces, Byte stream I/O and I/O conversions in Cedar*, stored as IODoc.tioga. IO is a large interface that contains three major components in addition to the generic operations on streams: procedures for creating streams from a few important data types, procedures for reading from a stream and performing input conversion, and procedures performing output conversion and writing the results to a stream.

[G18] Willie-Sue Orr, *How To Use Walnut*, stored as WalnutDoc.tioga.

[G19] Tim Diebert, *The Cedar Archive System*, stored as [Cyan]<Archivist>Documentation>ArchiveDoc.tioga.

## 3.2 Cedar Language References

[L1] *Mesa Language Manual*, Office Systems Division, Draft Version 11.0, March, 1984. The Mesa documentation is fragmented across this and the next reference, so you need to be familiar with both of them.

Previously, we used Maybury, and Sweet, *Mesa 5.0 Manual*, CSL-79-3, April, 1979, but this is out of print. The update to Mesa 6.0 was SDD, *Mesa 6.0 Compiler Update*, [Ivy]<Mesa>Doc>Compiler60.press.

[L2] Satterthwaite *et al.*, *Cedar Mesa 6T5*, [Indigo]<CedarDocs>Lang>Cedar6T5.press. This is a detailed description of the Cedar language, assuming one knows Mesa. Some changes since this document are enumerated in a shorter document, Documentation>Cedar7T11.press.

[L3] Horning, J., *Cedar Language Overview*, OverviewDoc.tioga; Lampson, B., *A Description of the Cedar Language: A Cedar Language Reference Manual*, CSL-83-15. This is a CSL "Blue and White".; Mitchell, J., *Annotated Cedar Examples*, CedarExamplesDoc.tioga.

[L4] Mitchell, J. *Stylizing Cedar*, CedarProgramStyle.tioga, *Cedar Style Sheet*, stylesheet.sil, .press.