D-machine 3.0 Extensions and DVT interface

Alexander Peyser

September 27, 2011

Contents

1 Introduction

The 3.0 version of the d-machine introduces a fundamental shift from a single Mac OS system to a clustered Unix approach. This has demanded a significant rewrite of the underlying operating system interface and an extension into the world of tcp connectivity. The 'dvt,' which previously was both an interface and a processor, has been replaced with a dedicated interface and 'dnode' manager. The dnodes handle all heavy processing independently, and can be viewed as threads within the larger d-machine.

From an interface perspective, the 'eye' and 'hand' have been translated from a Mac OS-centric system handled directly by dvt d-code to an X-windows based system. The eye is still handled directly by the d-code, but has been extended into a group of eyes, one for each dvt and dnode, functioning independently. The hand is now composed of an emacs frame combining emacs lisp code and d-code in the dvt and the dnodes, for which it acts as an agent.

Additionally, interfaces that will be discussed are the DVT macro bar, 'TheHorses' and the Tiny Window Manager (twm) elements of the interface. In term of a high-level model, the d-machine has been distributed over several machines and tools external to the d-machine proper, to handle the additional complexity due to clustering and to re-use open source code instead of recoding several internal elements.

A separate manual, "Emacs & D-machine: d source files and dvt" [Peyser, 2004], stored as `dvt-emacs.tex` and `dvt-emacs.pdf`, covers the details specifically of the emacs interface to the dvt. This manual will focus on overall user function and relevant d-code, integrating the usage of the system.

NB: On G5 Panther machines, all references in this manual to the Alt key must be replaced with the Apple key.

2 System Overview

All primary source code resides in `/mnt/Lab1/dm-3.0/`, which at the local UM cluster is a shared NSF mount. For other installations, it can be on a local machine which is rsync'd for updates. To build an architecture specific d-machine, `make` is called with one of the following paramters: `osx` for osx G5's, `g4` for linux G4's, or `g5` for linux G5's (which is not yet implemented). This builds a directory `/mnt/Lab1/dm-3.0/arch/` and a link to it in `/mnt/Lab1/dm-3.0-arch/`, where arch is osx, g4 or g5.

Inside this directory, all d-scripts, bash-scripts, perl-scripts and lisp code is inserted, and the executables `dvt` and `dnode` are placed. This directory (`/mnt/Lab1/dm-3.0-arch/`) should be your current directory when you run any d-machine associated code.

2.1 Environment

The d-machine can be built for linux running on a powerpc (g4), on a g5 when the 64-bit compiler is production ready, and under osx on a g5. The code is almost identical under all these architectures; osx, being a version of BSD unix, has a few parameter changes, in addition to some Apple 'tweaks,' to be kind, in precompiliation which had to be worked around. Emacsen under which the dvt works include versions 21.3.1 and 21.3.50.1. Each version uses optimized compilation flags, benchmarked primarily for double-precision floating point operations.

Under both linux and osx, dvt, dnodes and emacs use X-windows as the graphics server, managed by twm. Under linux, X is started by `startx`, and on osx the DarwinX button on the doc is used to start it. For the latter case there is the option of rootless or not. When not rootless, the X windows run directly over aqua; in the other case, an X root window replaces the aqua windows, and can be switched out with `Ctrl-Apple-A`. Either will work, to preference. However, the X root window does not interface well with the aqua screen saver.

Tiny Window Manager puts the buttons and borders around the windows. This should be part of the initial installation and is customized for d. The d windows have a class of "d_machine," which twm's `system.twmrc` recognizes. An alternate icon manager is used to collect those windows, and is initially located in the upper right-hand side of the screen. This 'docks' the windows for minimization and maximization, and groups d-machine window separately from other X windows.

2.2 DVT

The dvt proper exists as a child process of emacs, running under X. After initializing X and emacs, the dvt is started via the command `Alt-x dvt`. See the dvt-emacs manual [Peyser, 2004] for a general description of the interface. The dvt begins by loading `dvt_startup.d`, which initalizes the `userdict`, and begins the main interface loop, which is partially implemented in d. During installation, the lisp code is linked to begin the dvt in the proper directory.

Minimal d-code should be run directly in the dvt, for several reasons:

1. The memory allocated, including stacks sizes, is hard-coded in c. To run large jobs, the code must be recompiled. Dnodes have flexible memory allocation.

2. Debugging capabilities are severely limited. In order to minimize loss of interface under an error, whenever an `abort` event occurs, the stacks and dictionaries are cleared, returning to the main interface loop. This leaves little evidence of the cause of the error (there is a work-around though, the `debug_abort` mechanism in `dvt_startup.d`).

3. Running code in the dvt will interfere with management of d-nodes. Simply, the dvt may not be available for handling dnode communication while it is processing, and its behaviour may be quite complicated with numerous execution stack subsets running. This includes TheHorses functionality, and the macro bar.

4. The dnodes can be combined to multithread operations; the dvt shoudle exist, by definition, as a single master thread.

When the dvt starts, the following windows should pop up:

TheEye is the X version of the old eye. It allows file operations, and dictionary, list and array investigations. It is a purely mouse driven interface. On the top bar, the node to which it belongs is labeled (at first you only have the dvt eye).

TheHorses contain all currently connected nodes of the d-machine. Initially, it only contains the dvt itself (labeled `dvt`). This interface controls the current keyboard owner and reflects the busy state of each node. This is also mouse driven.

DVT macros allows the integration of mouse-driven and keyboard-driven events. One can choose a macro, which appears in the emacs dvt window as a d-command. This d-command may look at selections in the eye to open files, etc...

d_machine Icon Manager organizes the icons for the preceeding windows. This does not include the main emacs window which does not, by default, belong to the proper class.

2.3 Dnodes

Processing occurs in dnodes, each of which can act as a separate thread of operation, cooperatively or not. The can exist on the dvt's host machine, or on any other available machine. Currently, this has only been implemented for ppc's, but no obvious endian issues exist for intercommunication. The dnode host just needs to have the X libraries required to communicate with the host, but it does not need to be running an Xserver, itselve.

Each dnode runs as a server on a tcp port (5000+n). A script exists in the architecture directory: `dnode.sh`. It has one parameter (n), and start the dnode immediately. If the dnode fails because the port is currently unavailable, the script will continue trying to start the dnode at some small interval. This may occur after a dnode dies and before the os has finished the cleanup of the port connection, making the available once again.

The dnode itself will respond to a `HUP`, `QUIT` or `TERM` signal (such as by: `killall -HUP dnode`) by shutting down cleanly, and responds to an `INT` with an abort. In the former cases, the dnode.sh script will terminate immediately and not try to restart the dnode.

When initially started, the dnode has a 'tiny' memory, just enough to bootstrap itself. In this state, it uses minimal resources, including cpu utilization.

So, to summarize:

Local Start

1. open an xterm (terminal window)
2. change to your directory: `cd /mnt/Lab1/dm-3.0-arch/`.
3. run `dnode.sh`: `./dnode.sh 0` (or n).

SSH Start

1. open an xterm
2. make sure that you can access your local xserver, either with xhost +otherserver, or using ssh mechanisms (see section 3).
3. `ssh otherserver`.
4. `cd /mnt/Lab1/dm-3.0-arch/`
5. `./dnode.sh 0` (or n).

3 Running a d-machine

Once the dvt and dnodes are started, the machine can be integrated. The basic command for this is:

```
host port group memorysize _csu
```

host is a string identifying the dnode host machine. It should be of the form: `(klutz3.nonnerlabe)`. For G5's running OSX Panther, use `(klutz3)` with no domain appended. The name (localhost) generally does not work.

port is the port number offset for that host (n), usually a small number like 0,1,10..., depending on the number of dnodes running on that host.

group is a small negative number that puts several nodes into a 'group' with that same number. This is a device for sending the same command to multiple nodes.

memorysize is an array(5) of longs describing memory allocation as follows:

1: nopds The size of the operand stack. Between 1000 and 100000.

2: ndicts The size of the dictionary stack. Between 100 and 10000.

3: nexecs The size of the execution stack. Between 50 and 5000.

4: VM The size of the VM in megabytes. Between 1 and 1000.

6: userdictsize The size of the initial userdict. Between 200 and 5000.

At the dnode, `_csu` calls `vmresize`, which takes the final parameter of `_csu` as its only parameter. It creates the memory, and then `_csu` loads `startup_dnode.d`. This initializes the userdict for the dnode.

This file also initializes the X-connection to the dvt. An eye for the dnode is popped up on the Xserver identified by `/getmyname` at the dvt. This is initialized by default to the value of the environmental variable `$DISPLAY`,or the local hostname if that is set to localhost:x, where x is the display number. Usually, this is all set automagically. However, this can be a security issue; this means that `xhost +` may have to be called on the command line, or `/getmyname` may need to be set to an ssh tunnel by hand, before calling `_csu`.

At this point, several graphical changes should occur:

- TheHorses should be populated with the new dnode. A new line should appear saying something like klutz3:0, for a dnode on the klutz3 on port 0.
- A new eye should appear labeled in the same way.

This can be repeated to connect several dnodes. In order to disconnect from a dnode, call:

`host _dc`,

where host is the dnodes number in order of TheHorses interface. In other words, the first dnode you connect to will be 1, the next 2, ... If you attempt to reconnect later, empty spaces will be filled, and the number will be of its *position*, not the order in which it was created. When a node is disconnected, its eye will continue on the current screen. It can, however, be destroyed by calling `null vmresize` directly on the dnode before disconnection.

A dnode can be reconnected after a `_csu`, `_dc` sequences without reinitializing the memory by calling:

`hostname port group _c`,

where hostname, port and group are the same as in `_csu`. This can be particularly useful if the dvt dies. Operations on the dnodes are not interrupted. They continue, and any output goes to their standard error stream (probably the xterm on which they were started). By starting a new dvt, and using `_c`, the output is redirected to the new dvt without interrupting pending operations. However, this will not move the node's eye from its current screen.

3.1 Using TheHorses

To talk with the a dnode, TheHorses is the primary controlling interface. By left-clicking on a dnode or the dvt, it is selected as the primary keyboard owner. By clicking with button three (or `Ctrl`-button 1), the node and other nodes in its group are selected as the keyboard owner. All selected nodes appear in bold blue text. An alternate method for selected a node is via:

`target _t`,

where target is a node#, or group# (as in `_dc`, section 2.3).

While selected as the keyboard owner, all commands entered through emacs are directed to those nodes. To send a command to the dvt, it must be reselected as the

keyboard owner. The command is sent to the selected nodes when return is hit; if a command is entered, and then the node selected is changed, and finally return is hit from the keyboard, it will be sent to the newly selected node.

While a node is processing the sent command, the node is highlighted in TheHorses. If it is selected as the current keyboard owner, commands will be rejected when return is hit unless prepended by an exclamation mark (`!`). See the other manual for control keys associated with this.

When the operation is finished, the node will return to its unhighlighted state. This lock mechanism is intended to avoid accidentally interrupting a long-running process. If an error occurs at the dnode, `!stop` sent to the node will usually end the process, and set the node as ready. To force a node to appear to be ready, either click the node with button 2, or shift-button1, or call `target setready` in the dictionary `dvt`. This will clear the busy flag, but will not actually send any command to the node. It simply allows one to send a command without prepending a `!`.

These operations are summarized in section 5.1.

TheHorses belongs to the dvt. Its operations occur within the dvt, and do not 'speak' directly to the dnodes. They only change the state of the dvt, and thereby control the directing of keyboard input to the proper node, subsequently.

3.2 Tiny Window Manager

All windows are managed by twm. The manager is fully documented by the man pages (accessed by typing in emacs `Alt-x man` and then `twm`). However, a brief comment may be useful.

The d-machine windows, excluding emacs, will appear in the d_machine Icon Manager, which is initially on the upper right hand side of the screen. Each row has the name of the associated window, and clicking on it either minimizes or restores that window. Windows can be dragged by the title bar. On the right side of the window is a resize symbol. By dragging it past the edge of the window, one can resize windows; however, most d-machine windows are not resizable.

The first symbol on the left-hand side of the title bar will minimize/restore the window upon clicking. The next-symbol from the left will pop-up a bar with the options: Iconify (minimize/restore), Delete (request that the owning process kill the window), and Kill (immediately remove the window). Using kill will generally also kill the associated process (kill your dnode).

Windows pop up randomly on the screen. You will have to place them in some useful/rational order by dragging the title bars.

3.3 TheEye

Now that the dnodes are connected and can be operated on via the keyboard, TheEye can be analyzed. One exists for each dnode, in addition to the dvt. It is important to note that when an operation is done on an eye, a piece of d-code is placed on top of the execution stack, interrupting current operation. This has two implications:

1. While a d operation is within an operator (i.e., within a piece of c), TheEye will not respond until after its completion.
2. At the end of an operation, TheEye's operation will begin with the current stack, dictionaries, etc. . . , unaltered below the level of the current operation. Generally this is safe, as TheEye operations encapsulate themselves, leaving the state of the node/dvt unchanged. However, don't push it too far; clicking quickly during important/time-consuming operations is a recipe for disaster.

The eye is composed of:

Left Pane: stores complex objects that have been selected. Left-clicking puts identifying characteristics in the top bar. `Alt`-click, or middle-click selects the object for macro operations (it turns red). `Ctrl`-click, or right-click, opens the composite object in the right pane. `Shift`-click removes the object from the left pane. Userdict and the filesystem root (`/`) appear automatically in the left pane, and cannot be removed.

Right Pane: displays all the members of the last opened composite object. Left-clicking puts identifying characteristics in the top bar. `Ctrl`-click or right-click will place a composite object in the left-pane, and open it in the right pane. `Alt`-click or middle-click will toggle the select state for the object for macro operations (it will turn bold blue or return to black). Multiple objects can concurrently be selected in the right pane.

Bottom Bar: controls scrolling for the left and right panes; it is repeated twice, each one controlling the pane above it. The single-headed arrows scroll the window one line in the obvious direction. The double headed arrows scroll by

a page. The double headed arrow with a bar moves to the beginning or end of the pane. `Shift`-click or right-click will move proportionally, in terms of your position on the bar. In other words, right-clicking with the mouse in the middle of the bar will scroll to the middle of the pane, and right-clicking two-thirds to the right will scroll to two-thirds of the way through the pane.

Top Bar: displays identifying information of a d-object/file which has been left-clicked.

file: size, modification time, and permission.

directory: canonical path.

dictionary: total elements, and current number of elements.

list: elements.

array: type and size. For byte arrays, the initial string is also shown.

boolean,number: type and value.

The shortcuts are summarized in section 5.2.

Remember, each eye belongs to a separate node, which may be running on a different machine. The dvt is not directly linked to the eyes of the dnodes (only its own). Your X-server is displaying it on your screen, but the code that creates the window is running inside the process of the associated dnode. So, to access the state of an eye, you must run code in the associated dnode. This is important for the interaction of the several windows, particularly the macro interface (discussed in section 3.5).

3.4 Keyboard/Emacs

Of course, the primary interface to the system is via the command line, embedded within emacs [Peyser, 2004]. Once the connections have been made, TheHorse, TheEye and the command line interact to manage the operation of the entire d-machine.

A command entered if font-lock is on is initially colored by d-code conventions. After `return` is hit, this is converted by emacs to blue. Output from a node is by default in black text. In particular, output from the dvt is always black. Errors from any node are in red. This is encoded by the node using ansi color codes, and requires for conversion into actual colors that `ansi-color-for-comint-term-on` be called in emacs for the dvt buffer. That command is called by default by the dvt-mode code.

A line entered is understood in terms of a preceding tag by the dvt, which then feeds the phrase up to the newline, in the proper context, to the current target (which may be itself):

untagged the phrase is sent to the keyboard owner, to be interpreted as d-code.

`!` the phrase is decode to be sent to the keyboard owner, regardless of the current busy state.

`$` the phrase is a shell command to be executed by the keyboard owner.

`#` the phrase is interpreted as d-code by the dvt, regardless of owner. The string produced is then sent to the keyboard owner, to be executed as a shell command.

`@` same as `#`, but it is interpreted by the keyboard owner rather than the dvt.

For `#` and `@`, the command line should be of the following form:

`(string) fax` appends a shell string literal.

`faxLpage` appends the object selected in the left pane of the eye.

`faxRpage` appends all selected objects in the right pane of the eye.

A string-buffer and index are placed on the stack before the operation is called. Afterwards, `0 exch getinterval` is called and the resulting string is executed as a shell command. For `#`, this is done in the context of the dvt's eye, and for `@` this is in term of the keyboard owners eye.

In the dvt the shell command is spun off as a background process, allowing the continued operation of the dvt. However, in a dnode it is a foreground process, halting operation of the dnode until it has terminated. Generally, operating system commands are most usefully done directly from the dvt. The dvt and the dnodes are generally setup to have parallel file structures; in other words, the `/mnt/Lab*` are the same directories, either through nsf network mounting or by having the dnode and the dvt on the same local machine.

For normal d-code, the following operators are useful:

`getLpage` returns the object selected in the left pane of the eye.

`getRpage` returns a list of objects selected in the right pane of the eye.

These commands of course occur in the context of the current keyboard owner.

Further color-coding is available in the module `/mnt/Lab1/dm-3.0-arch/color.d`. This modules creates two operators in the userdict, `color_fax` and `color_text`, useful for adding color control codes to strings. They work both in the dvt and in dnodes. Usage instructions are embedded in the module's commentary.

3.4.1 Color coding of d-node output

To have alternate colors for the output of each node, a command line interface has been added to the dvt. The command `color_node` in userdict takes a color list and a node# (in terms of TheHorses), and sets the color output of that node. The coloring is actually done in the dnode.

For example:

```
[/cyan /italic] 1 color_node
```

will set output all output from a `/toconsole` command on node 1 to cyan and italic.

To remove coloring:

```
[/reset] 1 color_node
```

An alternate formulation for multiple nodes is:
`[[/color_name...]...] color_nodes`. This will call `color_node` on each node from 1...n, where n is the length of the list, with the respective color name list. Zero member color lists will cause the command to skip the associated node (not call `color_node` on that dnode).

The full list of color names are:

Foreground: `/black`, `/red`, `/green`, `/yellow`, `/blue`, `/magenta`, `/cyan`, `/white`.

Background: `/on_black`, `/on_red`, `/on_green`, `/on_yellow`, `/on_blue`, `/on_magenta`, `/on_cyan`, `/on_white`.

Styles: `/bold`, `/faint`, `/italic`, `/underlined`, `/slow_blink`, `/rapid_blink`, `/negative`. (Some of these are implemented fairly poorly by emacs).

Additionally, all of these operators will add a code the TheHorse, following the hostname and port number. The letter B followed by a single letter indicates the background color (**b**lack, **r**ed, **g**reen, **y**ellow, b**l**ue, **m**agenta, **c**yan, **w**hite). The letters will be in a color similar to that produced on emacs. The letter F followed by the same letter indicates the foreground color. Finally, the letter S followed by a code indicates the style of the output (**i**talic, **b**old, **f**aint, _ underlined, **s**low blink, **r**apid blink, **n**egative), and will be styled in a way suggestive of the output.

3.5 DVT Macro

An essential integrative window is the DVT macro bar. This bar runs in the dvt proper, and sends to the emacs screen a string that can be entered by the user to do a standard operation. Generally, this means that you click on a command, choose something in the eye and then hit `up-cursor return` or the `f1` key to send that string back to the dvt [Peyser, 2004].

The proper keyboard owner and matching eye must be used for many of these command. For example, to load a module one would click on *Load* on the macro bar. The command:

```
getRpage dup 0 get exch 1 get { exch dup 3 -1 roll fromfiles } forall pop
```

will appear on the emacs command line. Select all modules to load in the right pane of the dnode you want, and select that dnode as the keyboard owner. Then hit `f1` or `up-cursor return` to enter that command and thereby send it to the dnode. While the operation is occurring you will see that dnode highlighted in TheHorses.

The operations functional under osx are:

ChangeDir changes the current working directory. By default, the node runs in `/mnt/Lab1/dm-3.0-arch`, and all operations are relative to that. This command uses the currently selected directory in the left pane of the current keyboard owners.

MakeDir creates a directory. The selected directory in the left pane of the dvt eye is recreated on the current keyboard owners' host machines.

Remove removes all files or directories selected in the right pane of the dvt from the current keyboard owners' hosts.

Copy copies all files or directories selected in the right pane of the dvt to the directory selected on the left pane of the dvt, on the current keyboard owners' hosts.

Save saves objects to the file chosen in the right pane, using the object generator selected in the left pane. A literal directory path must be prepended by hand, such as `(/mnt/Lab3/ic1/)`. This all occurs in the context of the current keyboard owners. (I haven't actually seen this one used.)

Load loads modules from the files selected on the right pane. This also occurs in the context of current keyboard owners.

Additionally, under linux the following operators are available:

CD:

Mount mounts the cd drive to `/mnt/cdrom`.

Eject ejects the cd drive.

Stick:

Mount mounts the the stick to `/mnt/usbmem`.

Unmount unmounts the stick (which actually syncs the buffer).

Printer Ops are divided into two groups, *PrintFrom* and *PrintTo*. The operations in the first group print out the beginning of a print command, without a following newline. The second set completes the command, and adds a newline. So conceptually, the first group identifies the source type, and the second the destination. They print out the files currently selected in the right pane of the dvt, and generally are best used with the dvt as the keyboard owner. The *PrintFrom* options are:

Talk Directly connect to the print device. An xterm is created, which allows one to enter postscript commands to be executed by the printer or ghostscript.

Ascii Send ascii files transformed by the shell command `a2ps` into postscript to the print device. That command nicely formats some known file types, like c code, and prints out in a two-page per physical page format.

PS Send standard postscript directly to the print device.

xPS Prepends the reporter-created postscript files with standard include files (`/mnt/Lab1/ps/{form_2.ps,text.ps,graf_1.ps,struct.ps}`) before sending them to the print device.

The *PrintTo* options are:

gs Print to the ghostscript engine. This will pop up an xterm and a ghostscript window which will render the output. Hitting return will display page by page of the output.

LW Print to `LaserWriter Pro 630`. The printer has to be manually reset (hit the power) often.

xpdf Convert postscript to pdf (`file.ps` to `file.pdf`), and display the pdf with xpdf.

Finally, *LWstatus* prints to the dvt terminal the current status of the laser writer (to check whether you need to hit the power button on it).

Additionally, if the module `color.d` is loaded into the dvt, the macro commands will not be in the dvt's standard black, but will appear in bold blue.

4 Operator changes

Some operators behave differently than in earlier implementations, or have been added. These changes focus on tcp connectivity and the multithreaded behavior of the d-machine.

4.1 Operators `save` and `restore`

In earlier versions, `restore`'ing an uncapped `save` would result in the operand and dictionary stacks returning to their state at the time of the original save. In the current implementation, this often lead to segment violations. Therefore, the behavior of the operator has been changed.

Now, the `restore` operator on an uncapped `save` acts in a manner similar to a capped `save`. In other words, the memory in the box is deallocated without reverting the dictionary and operand stacks. However, the `restore` on a capped `save` will fail if either of those stacks contain reference into that region of memory (such as an

array on the stack which lives in the box). On the other hand, the uncapped save will simply remove the offending frames from the operand, dictionary and executable stacks.

4.2 Operators `send`, `console`, and `connect`

A `send` operator has been added, which sends operators from the dvt to the dnode and back. Its parameters are as follows:

```
socket [root (string)] |send| --
socket (string)        |send| --
```

where socket is a null object encapsulating a socket number as produced by `connect`, root is a list (including procedure), array or dictionary, and string is a piece of d code.

The (string) is executed on the foreign node. In the first form, the root object is placed on the operand stack before executing the (string).

Normally, the socket is retrieved on the dvt form `nodelist` in the `dvt` dictionary (see the description of `nodelist` in `startup_dvt.d`). This value, however, is originally created by the `connect` operator:

```
(servername) port# |connect| socket
```

To simplify coding of dnode code, the `console` operator has been added, which returns the socket of the dvt managing the dnode. This is primarily for use in send, where most operation have a return send to the dvt embedded in the command to be executed on the dnode.

For example, the following:

```
somenode (2 2 add _ console \(1 1 add _ \) send) send
```

will add 2 and 2 on somenode, then send a command back to the dvt to add 1 and 1.

4.3 Tilde'd operators

Tilde'd operators have been added to simplify the creation of procedures on the fly to send to dnodes or return to the dvt. A name that has been tilded on the execution

stack is neither placed on the operand stack as a passive name, or executed on the execution stack. Instead it is placed as an *active* name on the *operand* stack. Likewise, a tilded mark creates an active list (a procedure), of the elements between the mark and the close mark.

For example:

```
/somedict 2 dict dup begin
  /a {(a1\n) toconsole} def
  /b {(b1\n) toconsole} def
end def
/a {(a2\n) toconsole} def
/b {(b2\n) toconsole} def
/test ~[ somedict ~begin a ~b ~end] def
/somedict 2 dict dup begin
 /a {(a3\n) toconsole} def
 /b {(b3\n) toconsole} def
end
test
```

will output `a2` during the definition of `test`, and `b1` when `test` is called. The operators `a` and `somedict` are resolved and executed during the definition of `test`, but `b` is resolved and executed during the execution of `test`. Additionally, the resolution of `b` is in context of the `somedict` resolved during the definition, instead of the `somedict` that would normally be resolved during execution.

This can be most useful in locutions involving `send` (subsection 4.2), as follows:

```
/n 1 def
somenode [~[~nodedict ~begin n ~n ~add ~_ ~pop] (exec)] send
```

This will add the current value of `n` in the dvt (1), with the value of n in nodedict on the dnode (`somenode`), and display it.

5 Shortcuts

Most graphical commands can be entered in one of three ways: by a piece of dvt code, by a one-button mouse combined with control, alt, or shift, or by a three-button mouse click.

NB: On G5 Panther machines, all references to Alt key must be replaced with the Apple key.

5.1 TheHorses shortcuts

Operation	DVT command	1-button	3-button
Select Node	`target _t`	Click	1-click
Clear Busy	`target setready`	Shift	2-click
Select Group	`-target _t`	Ctrl	3-click

5.2 TheEye shortcuts

Left Pane:

Operation	1-button	3-button
Get Info	Click	1-click
Select	Alt	2-click
Open	Ctrl	3-click
Remove	Shift	Shift-1-click

Right Pane:

Operation	1-button	3-button
Get Info	Click	1-click
Select	Alt	2-click
Open	Ctrl	3-click

Bottom Bar:

Operation	Location	1-button	3-button
Scroll 1 line	$<$ or $>$	click	1-click
Scroll 1 page	$\ll$ or $\gg$	click	1-click
Scroll to end	$\mid\ll$ or $\gg\mid$	click	1-click
Scroll proportionally	proportional position	Shift-click	3-click

5.3 Emacs shortcuts

These are described in Peyser [2004].

5.4 DVT macros

The macro bar functions by left-click or single-click. The text produced on the terminal then must be sent to the keyboard owner with an `up-cursor return` or by `f1`. The commands available are explained in section 3.5.

5.5 X-windows

Under aqua on osx and full-screen X, `Ctrl-Apple-A` will switch between the X and aqua desktops.

References

Alexander Peyser. *Emacs & D-machine: d source files and dvt*. University of Miami, `email:apeyser@umiami.edu`, 3rd edition, June 2004. File `dvt-emacs.pdf`, original: November 2003.