BQUOTE

By: Kelly Roach

INTRODUCTION

This package implements the "backquote convention" punctuation marks backquote "`", comma ",", and commaat ",@" used in constructing lists. To use, load <LISPUSERS>BQUOTE.DCOM. This package offers all the behaviour that <LISPUSERS>PQUOTE.DCOM has to offer. You should not load PQUOTE if you load BQUOTE.

This package has many nice features that Interlisp's nlambda BQUOTE "type-in" approach does not have. Certain unexpected breaks that are possible with Interlisp's nlambda BQUOTE do not occur with this package. Unlike Interlisp's BQUOTE, this package checks for uncaptured commas, guarding against possible user error. Backquotes in files, in breaks during reads, around dots, and around other backquotes are handled appropriately. Backquoted forms are prettyprinted for the user.

FINDING BACKQUOTE

Backquote is character 96. Normally, typing <LF> on the 1100 keyboard or typing <SAME> on the 1108 keyboard produces a line feed. Typing <SHIFT-LF> or <SHIFT-SAME> will produce the backquote character. BQUOTE makes the backquote character easier to access by switching the <LF> keyaction with the <SHIFT-LF> keyaction on the 1100 keyboard and switching the <SAME> keyaction with the <SHIFT-SAME> keyaction on the 1108 keyboard. When BQUOTE is loaded, typing <LF> or <SAME> will produce backquote and typing <SHIFT-LF>or <SHIFT-SAME> will produce line feed.

USING BACKQUOTE

The punctuation marks backquote "`", comma ",", and commaat ",@" are used to build lists. Backquote "`" is used just like quote "'" and quotes everything in an expression not preceded by a comma ",". Hence,

`(A ,B ,C) translates to (LIST 'A B C)

`(if ,C then ,F) translates to (LIST 'if C 'then F)

'(it is ,(DATE)) translates to (LIST 'it 'is (DATE))

The punctuation commaat ",@" can be used to enter a segment into a list, using APPEND. So we have,

'(A ,@B) translates to (APPEND '(A) B)

'(PROG ,V ,@B) translates to (APPEND (LIST 'PROG V) B)

'(,F ,@M ,L) translates to (APPEND (LIST F) M (LIST L))

You can also nest backquoted forms. Do (SETQ X '(Y)). Then,

'(1 ,(A '(,X) B)) translates to (LIST 1 (A '(,X) B))

'(1 '(2 ,,X)) translates to

'(1 '(2 ,,X)) evaluates to (1 '(2 ,(Y)))

The BQUOTE package represents forms like "'X", ",X", and ",@X" as (\BQUOTE X), (\COMMA X), and (\COMMAAT X). Avoid using the atoms \BQUOTE, \COMMA, and \COMMAAT directly.

To determine which commas are captured by which backquotes, think

of your list expression as a tree,

"(A ,@B ,C . ,D)" has daughters "A", ",@B", ",C" and ",D"

",X" has daughter "X"

",@X" has daughter "X"

",,@,X" has great granddaughter "X"

On any path from the root node of the tree to a terminal node of the tree, backquotes capture commas just in the same way you would match left and right parentheses in a parenthesized expression. Work from the root node towards the terminal node. Some of the outermost backquotes may not have matching commas.

FONTS

GACHA fonts normally don't have a backquote char. The BQUOTE package inserts backquote chars into GACHA fonts and adjusts GACHA quote chars to be the mirror images of backquote chars.

READTABLES & PRETTYPRINTING

The BQUOTE package changes the readtable syntax of bakcquote "`" and comma "," to do the right thing in lisp readtables. Forms are added to PRETTYPRINTMACROS so that backqutoed expressions are prettyprinted back the way you typed them in. For this prettyprinting to work, SYSPRETTYFLG must be T (the BQUOTE package sets SYSPRETTYFLG when it is LOADed.) When SYSPRETTYFLG is NIL, backquoted forms will print out with atoms \BQUOTE, \COMMA, and \COMMAAT visible.

QUOTE CHAR

BQUOTE arranges for expressions like (QUOTE X) to print out as 'X. <LISP>LIBRARY>PQUOTE also offers this capability with respect to quote char. If you use the BQUOTE package, you don't need and you shouldn't use PQUOTE.

BREAKING

If for some reason you should break while a backquote form is being read in (say one of your own read macros didn't go as planned) the BQUOTE package will know how to handle this situation and not get confused. BQUOTE does this by adding a form to the system list BREAKRESETFORMS that resets BQUOTE before and after a break.

BQUOTE'S IMPLEMENTATION

The BQUOTE package uses CLISP's own mechanisms to store the translations of backquoted forms much as CLISP might store the translation of a FOR loop. Once a backquoted form is translated, it does not need to be translated again. Also like a FOR loop, when you compile a backquoted form, you actually compile the translation. The BQUOTE package is smart about the way it puts together its translations, so that there aren't unnecessary calls to CONS, LIST, APPEND, or NCONC.

COEXISTING WITH NON-BQUOTE USERS

Because BQUOTE modifies readtable syntax of characters quote "'", backquote "`", and comma ",", there will be some files created by people who don't use BQUOTE that cannot be properly LOADed unless there is a way to temporarily turn BQUOTE off. A BQUOTE user can turn BQUOTE off by doing

(BQUOTE.STOP)

and turn BQUOTE back on by doing

(BQUOTE.START)

To LOAD a file FILE that contains quotes, backquotes, or commas that aren't meant for BQUOTE, do

(BQUOTE.STOP)

(LOAD FILE)

(BQUOTE.START)

Fortunately, most files created by non-BQUOTE users can be LOADed safely without having to temporarily turn BQUOTE off.

THINGS TO KNOW

(1) The BQUOTE package uses backquote and quote punctuation not only on your terminal, but in your files. These files will not LOAD properly without the BQUOTE package loaded into your environment.

(2) The BQUOTE package smashes GACHA fonts a bit. In order for hardcopy to come out right, HELVETICA backquotes are substituted for GACHA backquotes in files.

(3) You should avoid using atoms \BQUOTE, \COMMA, and \COMMAAT.

(4) DEDIT and MKSTRING fail to prettyprint backquoted forms, but these failures can be lived with. (

5) The keyactions for <LF> and <SHIFT-LF> (or <SAME> and <SHIFT-SAME>) are switched.

CACHEFILES

By: Stan Lanning (Lanning.PA@Xerox)

What it does

Files that you specify are automatically copied onto a local disk when they are first read. Thereafter, any READ from that file will automatically use the cached copy of the file. Addtionaly, if the cacher cannot determine the validity of the cache due to the host being down, the user is notified and can indicate that the cached version should be used anyway.

How it works

The function OPENSTREAM is given a little advice. Specifically, the FILE argument is changed to be the name of the cached version of the file. This works for any code that uses the result of the OPENFILE or OPENSTREAM (OPENFILE calls OPENSTREAM) function as the file handle for further file access. For functions that think they know better that OPENFILE/OPENSTREAM this fails. The cached version of the file has the same version number as the "real" file, and this version number is used to determine cache validity. This is not a perfect technique, but it has the benifit that opening a file that has an explicit version number does not need to go to the file server to determine file validity.

How to use

Just load the (.DCOM) file. OPENSTREAM will be advised to use the function FileIShouldReallyOpen to compute the name of the cached file.

Variables you should know

These are all INITVARS, so you can set them up before you load the file.

CacheFiles -- an aList associating fileSpecs with pLists, where the pList defines how files that match the fileSpec should be cached. Examples of legal fileSpecs are *.DCOM, *.*, *.WIDTHS, <COLAB>LISP>*.*, or even *. The "*" character does not mean "any sequence of characters" as it does to the DIR command (sorry), it means "any name in this file name field is OK." The pList is a property list giving values for the properties Read, Write, Directory, and Versions (case is important!).

The Read property is used to determine if caching will occur on a Read. Legal values are T, NIL, DontUpdate or Ask: NIL means don't cache the file; T means do cache the file; DontUpdate will use a cached version if it is valid, but will not perform a cache update it is invalid; Ask will ask the user (in PROMPTWINDOW) whether to cache or not. Time out on this question is 30 seconds.

The Write property is used to determine if caching will occur on a Write. Legal values are T, NIL, or Ask. NIL means don't cache the file, T means do cache the file, and Ask will ask the user (in PROMPTWINDOW) whether to cache or not. Time out on this question is 30 seconds.

The Directory property specifies the directory (i.e. the local disk) that the file should be cached on. If NIL, the value of the variable DefaultCacheDirectory is used; if Ask you will be asked for the directory (in PROMPTWINDOW); otherwise it is taken the be the local directory to use for caching.

The Versions property defines the number of "extra" versions of a file to leave on a cache directory. A value of T means leave all old versions; NIL means to use the value of DefaultVersionsToKeep; a FIXP means to keep that many old versions.

The default value of CacheFiles is

```
((*.STRIKE Read T)
 (*.DISPLAYFONT Read T)
 (*.AC Read T)
 (FONTS.WIDTHS Read T)
 (LAFITE.* Read NIL)
 (*.MAIL Read NIL)
 (*.MAIL-LAFITE-TOC Read NIL)).
```

The Lafite stuff is in there because Lafite dies if it tries to use cached files. This was discovered when someone placed a (* Read Ask) at the end of CacheFiles.

Concerning font files: FONTCREATE looks for font files with INFILEP, and hence avoids using the cache. (However the cached version is used when the font file is loaded). If the host is down, INFILEP can take a long time to fail. One way around this is to do a (push DISPLAYFONTDIRECTORIES DefaultCacheDirectory) after loading CACHEFILES. FONTCREATE will then find the cached fonts directly (without going through the cacher), and will still cache fonts that are not stored locally. Since font files are seldom changed, this will work most of the time. One way to set all this up is to put (an edited version of) the following in your INITCOMS:

```
(* * File caching stuff)
(VARS (MyMachine "Curie"))
(FILES CacheFiles)
(P (if UseCache? then
       (push DISPLAYFONTDIRECTORIES DefaultCacheDirectory)
       (push INTERPRESSFONTDIRECTORIES DefaultCacheDirectory)
       (push PRESSFONTWIDTHSFILES (PACK* DefaultCacheDirectory
                                         (QUOTE FONTS.WIDTHS))))
```

If you are asked whether you wish to cache a file, and reply "NO", an entry of the form (file;* Read NIL) will be added to CacheFiles so you will not be asked again. If you answer "YES", an entry of the form (file;* Read T) is added. Similar things happen if you have a directory spec of Ask.

CacheSilently? -- if NIL, will write a message out in the prompt window when it is caching a file, otherwise no notice is given. Default value is NIL.

DefaultCacheDirectory -- see the description of CacheFiles above. The default value on Dorados and Dolphins is {DSK1}; on Dandelions it is {DSK}<LispFiles>.

DefaultVersionsToKeep -- the default number of "old" versions of a file to leave on the cache directory. The default value is zero.

KnownFileServers -- a list of names of file server hosts. Used to determine if a file is on a remote host or not. Default is (ERIS IVY INDIGO PHYLUM VAXC).

MyMachine -- not defined in the file, but see UseCache? below. Used to determine the default value of UseCache? and DefaultCacheDirectory.

UseCache? -- if NIL, then file caching is turned off. If UseCache? is T, then file caching is performed normally. If UseCache? is DontUpdate, then currently cached files will be used, but no updates to the cache will be made. Setting UseCache? to NIL provides a handy way of running if/when the file cacher gets in your way. The default value is determined by the function DefaultUseCache?, which returns T if the atom MyMachine is set and matches the value of (ETHERHOSTNAME NIL T) (e.g. the string "Curie"). If MyMachine is not set, or does not match the current machine name, you will be asked for the value of UseCache? and DefaultCacheDirectory. This provides an automatic way of disabling caching when using a "different" machine.

Problems/Restrictions

1) All file cache directories must be local disk partitions.

2) So far, only read caches work. Writes are not made to the cache directory.

3) Due to a problem with having a file open twice, it doesn't cache in a background process. When you first open a file that is not on the cache, or when the cached version is out of date and needs to be updated, there will be a delay while the file is copied.

4) If your cache directory does not have enough space for a copy of the file, a message is printed out in the prompt window and the file is not cached. Later, it may delete the oldest (i.e. least recently accessed) cache file.

5) A positive reponse to the question (HostUp? server) only means that server is echoing; there is no promise that server will respond to your requests.

Changes since last time

1) When a file is cached for read, it is given the same version number as that of the original file. The version number is now used to check cache validity.

2) The Directory property can now be Ask.

3) The Read property can now be DontUpdate.

4) Support for Xerox 1108s has been added.

If there are any questions/problems/complaints/requests, please let me know.

CALENDAR

By: Michel Denber (Denber.wbst @ Xerox.ARPA)

CALENDAR is a program which can be used to display a calendar on your screen, and keep track of events and appointments.

I. Starting CALENDAR

Load CALENDAR.DCOM from your favorite LispUsers directory and type (CALENDAR). You will get a menu of years (the menu always shows five years starting with last year). If you select a year, it will create a Year Window containing a calendar for that year. Each month in the Year Window is also a menu item. If you now select a particular month, CALENDAR will create a Month Window showing a calendar for that month. You can now point to a particular day within the month to bring up a Day Window.

CALENDAR uses the Prompt Window to display informative messages. You can SHRINK any of the CALENDAR windows to an appropriate icon when they are not needed.

II. Reminders

Initially the Day Window is empty except for an ADD menu item. If you select ADD, you will be prompted for a time and an event (or reminder) description. The time can be entered in almost any reasonable format, e.g., 2:30 PM, 2:30 P.M., or 1430.

The event cannot exceed one line in length (when you type a CR it assumes you're done). Your reminder is then displayed in the Day Window and stored on the active reminder list. The first few characters of the reminder will also appear in the Month Window. When the day and time of that reminder arrives, the program will let you know by displaying the reminder in the prompt window and flashing the screen.

The event-times for each reminder in the Day Window are also menu items which you can select if you want to specify a particular disposition of the reminder. The following options are currently available:

Flash (the default action) – The reminder is flashed in the prompt window. If you do not acknowledge it, it starts flashing the entire screen. You acknowledge the reminder (and cancel the flashing) by typing any key.

Beep - The reminder causes your Dandelion (or DO if you have the TI sound chip) to beep.

SendMail - The reminder is mailed to the recipient(s) of your choice.

Delete - Useful for deleting reminders that you no longer need. Normally reminders are deleted after they "fire."

Your active reminders are saved automatically in a file called CALREMINDERS in your connected directory when they are created. Thus you can save your reminders if you abandon your sysout or if your machine crashes (but see known bugs below).

The font used to display the Month Window is stored in CALFONT. You can change it for example by saying (SETQ CALFONT (FONTCREATE 'HELVETICA 18)). The change takes effect the next time you display a month.

III. Current limitations

Only one year, month, and day window can be shown at any given time.

If you reload CALENDAR on restarting Lisp and you have an active reminders file (CALREMINDERS), you have to reload CALREMINDERS manually.

CHATEMACS

By: Herb Jellinek (Jellinek.pa @ Xerox.ARPA)

ChatEmacs is a package that let you use your D-Machine mouse while CHATting to a Unix host running the Gosling (Unipress) Emacs text editor. Specifially, you can move dot by point, and scroll using the scroll bar.

To use ChatEmacs you must have the file dmouse.ml loaded into your Emacs.

(LOAD 'CHATEMACS.DCOM)

installs ChatEmacs. Call

(REMOVE.EMACS)

to remove ChatEmacs; call

(INSTALL.EMACS)

to re-install it.

You enable mouse movement of the Emacs "dot" by selecting Emacs on CHAT's middle-button menu, and turn it off the same way.

CHUNK-MENU

By: D. Austin Henderson, Jr. (A. Henderson.pa @ Xerox.ARPA)

CHUNK-MENU (and .DCOM) permit the creation and use of pop-up menus for long lists of items. The user sees one "chunk" of the list at a time. Menu items at the top and bottom allow him to move backward and forward in the list (to preceding or following chunks) and to "thumb" to a desired chunk via a menu of "first lines."

CHUNK.MENU.CREATE (ITEMS CHUNK.COUNT TITLE CENTERFLG MENUFONT ITEMWIDTH ITEMHEIGHT MENUBORDERSIZE MENUOUTLINESIZE) Creates a chunk menu for the menu items ITEMS. The chunks are menus with no more than CHUNK.COUNT items. The menus have menu characteristics given by the rest of the parameters (see the discussion of menus in the Interlisp-D Reference Manual). Returns the chunk menu.

CHUNK.MENU.MENU (CHUNK.MENU POSITION) Activates (pops-up) CHUNK.MENU centered at POSITION. If position is not specified, the position of the cursor is used. The user can select items, or move to other chunks by using the up, down, and thumbing entries. All chunks appear centered at POSITION or the original position of the cursor (prevents the menus from "walking" across the display. When an item is selected, the value returned is as specified for items of menus. CHUNK.MENU is modified so that the next activation will start in the chunk visible when the selction was made (it remembers which chunk the last selection was made in).

COLORUTILITIES

By: Ken Feuerman (Feuerman.pa @ Xerox.ARPA)

COLORUTILITIES.DCOM is a set of functions for the 1100 and 1132 which allow a user to create windows and menus on the color screen. This file should be loaded AFTER COLOR.DOCM has been loaded. Please see COLOR documentation for important information about the needed hardware for color.

Color Windows

Color Windows are not quite the same as their black and white counterparts. A Color Window is a window with certain properties that make its display appear on the color screen. By definition, they are transparent on the color screen; i.e., overlaying one window on another on the color screen will not obliterate the bottom window. Therefore, displays to color windows can get confusing if the windows are overlapping. When you create a color window, you are asked for a border color. This color becomes the "resident" color; i.e., all display operations to that window appear in that color unless specified otherwise. Also, the window menu for color windows (press the right button while in a window) appears in the color of the border color.

New Background Menu Commands

There are three new Background Menu Commands which allow for easy turning on and off of the color screen, the black and white screen, and switching the cursor between screens. The Background Menu is the menu that pops up whenever the right mouse button is pressed outside of any window. The commands are:

Switch: Switches the cursor from the black and white screen to the color screen, or vice versa.

Display On/Off: If the black and white screen is on, it is turned off. The color screen is turned on (if not already on) and the cursor is moved to it (if not already there). If the black and white screen is off, it is simply turned on.

Color Display On/Off: Same as Display On/Off, only for color screen.

When using the above menu commands, it is impossible to have both screens off at the same time or to have the cursor on a screen that is turned off.

The following are the functions to create color windows and display menus in color:

(COLORCREATEW Region Title Bordersize Bordercolor RedisplyWhenCloseFlg) -- Creates and returns a color window, and displays it on the color screen. If Region is NIL, the user is prompted for one as in CREATEW, only the prompting occurs on the color screen. Default Bordersize is 4, default Bordercolor is 15 (after loading COLORUTILITIES.DCOM, color 15 is WHITE).

RedisplayWhenCloseFlg indicates whether REDISPLAYW should be called when closing the window. This is very handy if the REPAINTFN for the window uses INVERT as its DSPOPERATION (INVERTing is more easily displayed and un-displayed when dealing with transparent windows). Its display will be wiped clean without destroying the contents of other perhaps overlapping windows. If RedisplayWhenCloseFlg is NIL, the user should make sure to clear the window before closing it [either manually or by specifying (WINDOWADDPROP WINDOW 'CLOSEFN (FUNCTION CLEARW))], otherwise its display will be left up on the screen. Note that when clearing a color window, any overlapping windows will have their overlapping regions destroyed. Selecting Redisplay on the right button color window menu will restore its border and call the window's REPAINTFN.

(MENU Menu Position ReleaseControlFlg) -- This is a redefinition of the standard MENU function. It works the same way as before if the cursor is on the black and white screen. If the cursor is on the color screen, the pop up menu appears on that screen. Position defaults are as before. ReleaseControlFlg is not implemented yet.

There are several ways to determine the color of an item in a menu. The first check is in the item itself. If in the list which describes an item there is a list whose CAR is the atom COLOR, the color of the item is retrieved from the CADR or CDR of that list, whichever happens to be there. If not, then the color of the item is the color of the menu outline. This is determined from a property of the menu (NOT a field). To enter the color of the menu outline, perform (PUTMENUPROP Menu 'MENUOUTLINECOLOR Color). The default color here is (MAXIMUMCOLOR). Example:

```
(SETQ COLLEGEMENU
        (create MENU
              ITEMS ← [LIST
        (LIST 'POMONA '(POMONA.FN) "Runs Pomona function"
                                   (LIST 'COLOR 'BLUE) )
        (LIST 'CMC '(CMC.FN) "Runs CMC function"
                                   (CONS 'COLOR 4) )
        (LIST 'HMC '(HMC.FN) "Runs HMC function") ] ) )
(PUTMENUPROP COLLEGEMENU 'MENUOUTLINECOLOR 'GREEN)
```

This will create a menu with a GREEN outline and with three items. POMONA will appear in BLUE, CMC will appear in color number 4, and HMC will be in GREEN (from the outline color).

The Menu argument is otherwise very much the same as black and white menus, except that some of the menu fields controlling display and such aren't implemented in color yet, so these fields are ignored for now. Those fields are:

WHENHELDFN – Currently nothing happens when a button is held.

WHENUNHELDFN – Similarly nothing.

MENUPOSITION – Works or operates as though this were NIL (regardless if this was actually specified).

MENUBORDERSIZE – Always treated as 0.

All other fields (ITEMS, WHENSELECTEDFN, MENUOFFSET, MENUFONT, TITLE, CENTERFLG, MENUROWS, MENUCOLUMNS, ITEMHEIGHT, ITEMWIDTH, MENUOUTLINESIZE, and CHANGEOFFSETFLG) are treated the same as they are normally treated. The same menu should be usable on both the monochrome and color screens.

(ADDMENU Menu Window Position) – – Only if Window is a color window is Menu made into a menu on the color screen that remains active in a window. If Window is NIL or is a normal window, this function operates the same as before with Color ignored. The same comments above about Menu apply. Returns the window to which Menu was added.

(COLORMENU Menu Position ReleaseControl Flg) – – Forces Menu to appear on the color screen.

(COLORADDMENU Menu Window Position) - - This time if, Window is NIL, a color window is created first and then the Menu is added. Behavior is unpredictable if Window is not a color window.

(COLOREDITBM Bitmap) - - Make sure that Bitmap has 4 bits per pixel (that it is in fact a color bitmap). This will open a color bitmap editor window on the color screen. Operations on this bitmap editor work pretty much the same as on the black and white screen with a few exceptions. Left button lights up a grid square in the currently selected color; middle resets it to its previous color; and right button erases it to black altogether. Moving the cursor up to the shaded area across the top and pressing either the left or middle button brings up a command menu with six options:

Reset - - Returns the bitmap in the grid to its state at the beginning of this edit session

Clear - - Clears the bitmap to black

Move - - Allows the user to specify which region to put in the grid if the bitmap is too large to fit in the grid.

Color - - Sets the color currently being drawn when pressing the left button. The shaded area across the top changes to match that color.

OK - - Saves changes when finished

Stop - - Deletes changes when finished

The returned value is the color bitmap that was edited.

Some other useful functions are:

(GETCOLOR) - - Puts a pop up menu on the color screen (make sure that the screen is on) and allows the user to select from 15 colors displayed in rectangular regions (color number 0 is left out); returns the color number of the color selected, or NIL if none is selected.

(GETCOLORREGION) - - Analogous to GETREGION, only the "ghost" region being swept out appears on the color screen; returns the resulting region.

(COLOR.CREATE.TTY Region Color) - - Creates a color window of color Color (default 15) on the color screen occupying Region (if NIL, Region is prompted for), and assigns that window as the top level typescript window; returns the old TTYDISPLAYSTREAM.

(COLORGETBOXPOSITION Width Height) -- Similar to GETBOXPOSITION, only the "ghost" region appears on the color screen; returns a POSITION in the coordinated of the color screen.

(COLORGETBOXREGION Width Height) -- Same as COLORGETBOXPOSITION, only it returns the region occupied by the box of width Width and height Height.

(COLORIZE.SCREEN.REGION Sourceregion Color TargetPosition Operation) -- Bits a region specified by Sourceregion directly onto the color screen with origin at TargetPosition. Operation determines how the bits on the color screen are merged.

(COLORIZE.WINDOW Window Color TargetPosition Operation) -- Puts an image (not an actual window) of Window on the color screen in color Color at TargetPosition using Operation.

(SWITCHSCREENFN) -- The function that is called by the Switch option off the Background Menu.

(SWITCHDISPLAYON/OFF) -- The function that is called by the Display On/Off option off the Background Menu.

(COLOR SWITCHDISPLAYON/OFF) -- The function that is called by the Color Display On/Off option off the Background Menu.

COMHACK

By: H. Thompson

COMHACK works by hacking the insides of DWIM so that before translating any IF or FOR expression, all comments are removed. The advice is in terms of a function called \StripComments, which can be used to accomplish similar modification to any other clisp word. The end result is that comments are allowed anywhere inside the affected form, and are never seen. This means that in the unlikely event that your code was actually working by using a comment for value, it will cease to work.

COMMENTS

By: Don Cohen (DonC @ ISIF.ARPA)

This is a documentation facility which allows the user to associate a list of comments with a word, file them and selectively retrieve them.

COMMENT [PROPERTY]
is the property under which documentation is stored. It is a list of comments for the word on whose property list the property resides. Each comment has an ID and DECRIPTION as documented in DC and a body, either a string or a file pointer. Comments are created by DC and displayed by DOC.

COMMENTS [FILEPKGCOM, FILEPKGTYPE]
The file package command (COMMENTS * FOOCOMMENTS) or (COMMENTS FOO1 FOO2 ...) dumps the documentation of the elements of FOOCOMMENTS or of FOO1, FOO2, etc. The corresponding file package type is also defined.

(DC WORD ID DESCRIPTOR) [NLAMBDA SPREAD]
DC creates a comment (documentation) for the specified word. Each word may have arbitrarily many comments, but only one for a given ID. When an old ID is used the comment for that ID is replaced. WORD and ID should be atomic. DESCRIPTOR is an arbitrary Lisp object (normally a list) that describes the comment. It can be used to decide whether the comment should be printed by DOC. Comments are stored under the COMMENT property of WORD. This can be edited if you want to delete comments, change their order or change their descriptor fields. The text of the comment is stored either as a string (when it is first typed in) or as a file pointer. As a default, file pointers are used whenever possible. They are created whenever a comment is written to a file or read from a file. These defaults may be overridden by setting the flag DC-DEFINE to store a string read from a file, and DC-RETAIN to keep a comment as a string when it is written to a file.

The following conventions are suggested for documentation:

> Every file should contain a comment for the file name which at least lists the other words with documentation in the file. This is just a way for a user to find out what documentation exists. Of course, if you want to describe the file while you're at it, so much the better.

It is expected that most words with documentation will have only one comment, and that doc ought to print it without asking. Therefore, the suggested usage (reinforced by the default DOCFILTER function) is that a comment ID of NIL (which is what you will get if you type "(DC FOO)") indicates that the comment should be displayed. For any other ID, the default DOCFILTER will ask the user.

(DOC word1 word2 ...) [NLAMBDA NOSPREAD]

DOC retrieves the documentation for the words specified. For each comment (in the order of the COMMENT property) DOC calls (DOCFILTER word id descr) where the parameters are the same as for DC. If the result is non-NIL the comment is printed. The default DOCFILTER prints its arguments and if the id is NIL returns T (i.e. print it). If the id is non-NIL it asks the user whether to print the comment. This is meant to reflect the common usage of DC to give a word only one comment by just doing (DC word).

OriginalFILERDTBL [VAR]

The comment package alters FILERDTBL. In particular, it uses <esc> to delimit comments. In case you want to read other files that may contain <esc> followed by 3 blanks followed by newline (or other files that do not conform to changes made to FILERDTBLE), it is handy to have the original FILERDTBL around to bind to FILERDTBL. If OriginalFileRdtbl is not bound when the comment package is loaded, it is given the value of FILERDTBL.

Problems

At the moment there is no good way to edit the text of a comment. One hack that works is to just edit the file with a text editor and then load it into lisp (which will rewrite the filemap. In the long run, the best plan seems to be not to write a text editor but provide an interface to the editor of your choice. Unfortunately, there are many text editors with different interfaces. At the moment I'm leaning toward picking a standard file name (doc.temp or some such), writing the comments to it, calling the editor of your choice with subsys and then reading the file back in with DC-DEFINE set to T. This means that the user must load the file and save it in addition to editing. Any suggestions are welcome.

At the moment the comment package contains a sleazy hack which enables comments to be copied onto a .COM file by TCOMPL. (The comment is transformed by a readmacro into an EVAL@COMPILE.) This is certainly not the "right" solution, but it works (for now).

LOADFNS seems to work all right for comments - remember that the pattern you want is (DC <word> ...). GETDEF and friends seem to work reasonably well (which is utterly amazing), but of course they tend to return comment pointers. This is what you want e.g. for editdef - it lets you reorder the comments, change their attributes etc. (One problem is that GETDEF from a file

tends to get other definitions besides the one you wanted. This is due to GETDEF's limited understanding of where to find definitions of types it doesn't know.)

The use of file pointers tends to lead to problems when MAKEFILE bombs out. If you ctl-D out of MAKEFILE, the comments will be correct, but you may have to do something to make the files accessible. In particular, you should close the file you were making, and undelete its former version if necessary. Otherwise, the file pointers will point to files that can't be found. After that you can redo the MAKEFILE.

COMPARETEXT

By: Michael Sannella (Sannella.pa @ Xerox)

COMPARETEXT is a rather non-standard text file comparison program which tries to address two problems: (1) the problem of detecting certain types of changes, such as detecting when a paragraph is moved to a different part of a document; and (2) the problem of showing the user what changes have been made in a document.

The text comparison algorithm is an adaptation of the one described in the article "A Technique for Isolating Differences Between Files" by Paul Heckel, in CACM, V21, #4, April 1978. The main idea is to break each of the two text files into "chunks" (words, lines, paragraphs, ...), hash each chunk into a hash value, and match up chunks with the same hash value in the two files. This method detects switching two chunks, or moving a chunk anywhere else in the document.

Two text files can be compared with the following function:

(COMPARETEXT NEWFILENAME OLDFILENAME HASH.TYPE GRAPH.REGION)

NEWFILENAME and OLDFILENAME are the names of the two files to compare. The order is not important, except that in the resulting graph the NEWFILENAME information will appear on the left, and the OLDFILENAME info on the right.

HASH.TYPE determines how "chunks" of text are defined; how fine-grained the comparison will be. This can be PARA to hash by paragraphs (delimited by two consecutive CRs), LINE to hash by lines (delimited by one CR), or WORD to hash words (delimited by any white space). HASH.TYPE = NIL defaults to PARA.

GRAPH.REGION is the region on the display screen used for the file comparison graph. If GRAPH.REGION = NIL, the system asks the user to specify a region. If GRAPH.REGION = T, a region in the lower left corner is used.

COMPARETEXT creates a graph with two columns. Each column contains the file name of one of the files, and lists the chunks from that file. Each chunk is represented by an atom NNN:MMM, where NNN is the file pointer of the beginning of the chunk within the file, and MMM is the length of the chunk. Lines are drawn from one column to the other to show which chunks in one file are the same as those in the other file. Chunks with no lines going to them do not exist in the other file. [Note: a series of chunks in one file which are the same as a series of

chunks in the other file are merged into one big chunk. A series of unconnected chunks is also merged.]

Pressing the LEFT mouse button over one of the chunk nodes causes the node to be boxed, and a Tedit window to be opened on the file, with the appropriate text selected. If a Tedit window to the file is already active, the selection is simply moved.

Pressing the MIDDLE mouse button over a chunk node raises a pop-up menu with the items: PARA, LINE, and WORD. If one of these is selected, COMPARETEXT is called to compare the selected chunk with the last selected chunk (the one that is boxed), using the hash type selected. COMPARETEXT creates a new graph window, using a region slightly offset from the region of its "parent" graph.

If the mouse is buttoned outside of the PARA/LINE/WORD menu, no comparison is done, but the selected node is boxed. The PARA/LINE/WORD menu is always brought up a little away from the cursor, so pressing double-MIDDLE-button over a chunk node is a way to change the boxed node without calling Tedit.

Important note: white space (space, tab, CR, LF) is used to delimit chunks, but is ignored when computing the hash value of a chunk. Therefore, if two paragraphs are identical except that one has a few extra CRs after it, they will be considered identical by COMPARETEXT.

Loads in: GRAPHER.DCOM

COMPILEFORMSLIST

By: Ron Kaplan (Kaplan.pa @ Xerox)

Compileformslist[var] compiles the list of forms bound to VAR as a single function, and then changes the binding of VAR so that it simply calls that function. The symbolic definition is saved on the property list of the function-name (which is gensymed) for future reference.

This function may be called repeatedly.

Example: Executing (COMPILEFORMSLIST 'AFTERSYSOUTFORMS) before a sysout can speed up subsequent entries into that sysout.

COMPMODEREC

By: Ron Kaplan (Kaplan.pa @ Xerox.ARPA)

This package enables COMPILEMODE-dependent record declarations to be declared so that access and layout for fields of a given record may be specified in a single declaration for different target implementations.

COMPREC is defined as a new record-defining word ala RECORD, HASHLINK, etc. A COMPREC "declaration" is of the form (COMPREC name . decls), where decls is an alist of (compilemode decl) pairs, where compilemode can be an atomic COMPILEMODE value or a list of such values ala SELECTQ.

Thus (COMPREC FOO (D (DATATYPE (A B))) (PDP-10 (BLOCKRECORD (A B]

COPYIMAGE

By: Marcel Schoppers (Schoppers.pa)

The COPYIMAGE package facilitates making hardcopy or PRESS output of either any window on a screen, or else the ENTIRE screen.

Loading COPYIMAGE updates the background menu. Then button one of the CopyImage sub-items of the background menu. If you ask to copy the whole screen you are done; if you ask to copy a window you will be prompted to button your chosen window. Hardcopy output goes to the FULLPRESS printer of your DEFAULTPRINTINGHOST.

These functions are also provided by the LOOPS icon, but people not using LOOPS can use this utility in INTERLISP.

The Hardcopy item in the standard background menu also does something similar, but is more like a SNAP: you have to define the corners of a box with the mouse. This is

1) not as fast to use

2) not as neat on output

3) not capable of copying a whole screen

4) not capable of making PRESS files.