PRETTYFILEINDEX

By: Bill van Melle (vanMelle.PA@Xerox.com)

INTRODUCTION

PRETTYFILEINDEX is a program for generating indexed listings for Lisp source files. PRETTYFILEINDEX operates by reading expressions from the file and reprettyprinting them to the output image stream, building up an index of the objects as it goes. The index is partitioned by type (e.g. FUNCTIONS, VARIABLES, MACROS, etc.); within each type, the objects are listed alphabetically by name along with the page number(s) on which their definitions appear in the listing.

PRETTYFILEINDEX also modifies the Exec's and the FileBrowser's SEE command to prettyprint the file being viewed, if it is a Lisp source file. It also modifies the PF and PF* commands to prettyprint the requested function body. Together, these features mean you can use the NEW & FAST options to MAKEFILE to speed up file creation without sacrificing the ability to get pretty listings or see the files prettily inside Lisp.

PRETTYFILEINDEX performs some additional niceties in the listing: it prints bitmaps by "displaying" them, rather than dumping their bits; it translates underscore to left arrow (for the benefit of Interlisp listings); it prints quote and backquote in a font in which they are clearly distinguishable; and it suppresses some of the "noise" in source files, such as the filemap.

The module also contains a function MULTIFILEINDEX that can be used to generate a merged index of items from a whole set of files being listed.

PRETTYFILEINDEX subsumes, and is incompatible with, the modules SINGLEFILEINDEX and PP-CODE-FILE. You can, however, load PRETTYFILEINDEX on top of either one, and it will successfully wrest control of LISTFILES from them. PRETTYFILEINDEX has several advantages over SINGLEFILEINDEX: the prettyprinter has fine control over positioning of the output stream, so things that are supposed to line up do, despite font changes and variable-width fonts; the entire page is used, rather than sacrificing the bottom quarter or so due to lack of control over page breaks; and the use of an image stream allows bitmaps to be rendered directly.

USING PRETTYFILEINDEX

For ordinary use, just load PRETTYFILEINDEX.LCOM. This redefines LISTFILES1 so that calling LISTFILES or using the File Browser's Hardcopy command invokes PRETTYFILEINDEX if the file is a Lisp source file. The listing is created by default in a single background process that handles all LISTFILES requests. The file being indexed needn't be loaded, or even noticed (in the File Manager sense) as long as the file's commands don't require customized prettyprinting defined by the file itself. The index is printed at the end of the listing; you are expected to manually transpose the index to the front of the collection of paper that emerges from the printer.

PRETTYFILEINDEX normally assumes that you are printing one-sided listings. However, if your global default is for two-sided (currently this means that EMPRESS#SIDES = 2) or you specified two-sided in the options you passed to LISTFILES, it will prepare the output as if for two-sided listing. For example, from an Interlisp exec,

```
(LISTFILES (SERVER "Perfector:" %#SIDES 2) FOOBAR)
```

causes the file FOOBAR to be listed two-sided on the print server Perfector: (the % is the Interlisp reader's escape character, needed to quote the special character #; in an XCL exec the escape character is \, and from other packages you also have to qualify the symbols LISTFILES, SERVER and #SIDES with the package prefix IL:).

For two-sided listings, the margins are symmetric, instead of being shifted a bit to the right, page numbers appear on the outside edge of the page, and a blank page is inserted at the end of the listing if necessary to ensure that the index starts on an odd page (and hence is transposable to the front).

PRETTYFILEINDEX prettyprints the file's contents and prints indexed names using the package and read table specified in the file's reader environment, which appears at the beginning of the file. It assumes, as does most of the file manager, that the reader environment is sufficient to read any expression on the file. If you have violated this assumption, for example, by referring in the file to a symbol in another package that is defined on a file that is indirectly loaded by the file somewhere in its coms, you will probably need to LOADFROM the file before you can list it.

INDEXING MULTIPLE FILES

Ordinarily, you list files and get one index per file. If a module is made up of several files, you may want a master index of the whole set of files, so that you don't have to remember which file contains a function, macro, etc. that you are looking up. This job is handled by MULTIFILEINDEX:

(MULTIFILEINDEX *files printoptions*) [Function]

> This function lists each of the files in the list *files* using PRETTYFILEINDEX and then produces a master index by merging all the individual indices. The master index is appended to the output of the last file listed. The argument *files* can be a list of file names and/or file patterns, such as "{FS:}<Carstairs>RED*", or a single such pattern. In the pattern, unless explicitly specified, the extension defaults to null and the version to "highest". The argument *printoptions* is a property-list of options, the same as the *printoptions* argument to SEND.FILE.TO.PRINTER or PRETTYFILEINDEX, with the addition of some options recognized by MULTIFILEINDEX, described further below.

As each file is listed, its pages are numbered with an ordinal file number plus the page number within the file; e.g., in the first file the pages are numbered 1-1, 1-2, ..., in the second file 2-1, 2-2, etc. The master index then refers to page numbers in this form, although each individual file's own index shows only the file-relative page numbers. Alternatively, you can tell MULTIFILEINDEX to number all the pages consecutively, rather than using "part numbers", by giving the option :CONSECUTIVE, value T in *printoptions*.

In the event that some files in the set have different reader environments, the master index is printed in the environment used by the majority of the files. More specifically, MULTIFILEINDEX independently chooses the package used by the majority of the files and the readtable used by the majority; in the case of a tie, the file later in the set wins. If this default is not adequate, you can specify the environment yourself by giving the :ENVIRONMENT option. The value should either be a reader environment object, such as produced by MAKE-READER-ENVIRONMENT, or a property list of the form used by the MAKEFILE-ENVIRONMENT property.

For example,

```
(MULTIFILEINDEX "<Barney>Rub*"
        '(:CONSECUTIVE T
          :ENVIRONMENT (:PACKAGE "JABBA" :READTABLE "XCL")))
```

would list each of the files matching "<Barney>Rub*.;", numbering the pages consecutively from the first file through the last, and printing the master index with respect to the package JABBA and read table XCL.

INCREMENTALLY REPRINTING MULTIPLE FILES

If you have used MULTIFILEINDEX to list a group of files, and later one of the files changes, or maybe the printer just ate part of your listing, you might want to update your listing without reprinting the entire set of files. You have two options.

(1) You can have PRETTYFILEINDEX reprint the one file that changed (or was eaten). Specify the print option `:PART` *n* to have it treat the single file as the *n*th part of a multiple listing, or the option `:FIRSTPAGE` *n* to have it start numbering the pages at *n* instead of 1 (for the case where you used the `:CONSECUTIVE` option to MULTFILEINDEX). For example,

```
(LISTFILES (:PART 3) "<Barney>Rubric")
```

would reprint `<Barney>Rubric` as the third file in a group. Of course, this doesn't reprint the master index, but it only has to process the one file, which may be adequate for your needs if things didn't move around too much.

(2) You can have MULTIFILEINDEX process the entire set of files again, but only print some of them. You specify this by parenthesizing the files you don't want printed. That is, each element of the *files* argument to MULTIFILEINDEX is a file name or a list of file name(s); those files inside sublists are processed but not printed. You cannot specify patterns. The master index is listed after the last file, as usual, except that if the last file was in a sublist, and hence not printed, the master index will appear as a separate listing. Calling MULTIFILEINDEX in this manner is nearly as computationally expensive as calling it to list the whole set for real (it omits only the transportation to the printer), but it does save paper and printer time.

LISTING COMMON LISP FILES

Ordinarily, PRETTYFILEINDEX only processes files produced by the Lisp File Manager; it passes all others off to the default hardcopy routines. However, you can tell it to process a plain Common Lisp text file by passing the print option `:COMMON`; e.g.,

```
(LISTFILES (:COMMON T) "conjugate.lisp")
```

PRETTYFILEINDEX still processes the file by reading and prettyprinting, just as for Lisp files. It starts in the default Common Lisp reading environment (package USER and read table LISP), and evaluates top-level package expressions, such as `in-package` and `import`, in order to continue reading correctly. The index is printed in whatever the environment was at the end of the file.

Of course, this is of fairly limited utility, as all read-time conditional syntax is lost: comments, #+, #o, etc. The one exception is that top-level semi-colon comments are preserved—they are copied to the output directly, rather than being read.

Customizing PRETTYFILEINDEX

The remainder of this document describes various ways in which PRETTYFILEINDEX can be customized.

HOW TO SPECIFY INDEXING TYPES

Initially, PRETTYFILEINDEX knows about most of the standard file manager types. In addition, it handles all the types defined by DEFDEFINER. For definers with a :NAME option, it assumes that the function is free of side effects. PRETTYFILEINDEX also notices (but does not evaluate) DEFDEFINERs that appear on the file it is currently indexing, which should appear before any instances of the type so defined in order for correct indexing to occur. Of course, it can't know about definer types that are defined on some other file unless you load it.

You can augment the set of indexing types, or override the default handling of definers, by adding elements to the following variable:

`*PFI-TYPES*` [Variable]

A list of entries describing types to be indexed and a way of testing whether an expression on the file is of the desired type. Each entry is a list of up to 4 elements of the form (*type dumpfn namefn ambiguous*), the first two of which are required:

type The name of the type, e.g., `MACRO`. This name will appear as the name of the index for this type, e.g., "MACRO INDEX". *type* is usually the name of a file package type, though it need not be. It must be a symbol.

dumpfn The name of the function that appears as the CAR of the form that defines objects of type *type* on the file, or a list of such names. E.g., for type `TEMPLATE` it is `SETTEMPLATE`; for type `VARIABLES` it is `(RPAQ RPAQQ RPAQ? ADDTOVAR)`.

namefn A function that tests whether the expression that starts with *dumpfn* really is of the desired type, and returns the name of the object defined in the expression. The function takes as arguments (*expr entry*), where *expr* is the expression whose CAR matched the entry. The *testfn* should return one of the following:

`NIL` the expression is not of the desired type.

name the expression defines a single object of this name and of the type given in the entry.

a list the value is either a single list or a list of lists, each of the form (*type . names*), meaning that the expression defines each of the names as having the specified type.

If the *namefn* is `NIL` or omitted, the name of the object is obtained from the second element of the expression. If that element is a list, the name is taken to be its CAR, or its CADR if the element is a quoted atom.

ambiguous True if the expression is ambiguous, in the sense that even if *namefn* returns a non-NIL value, it is possible for this expression to also satisfy other entries in `*PFI-TYPES*`. E.g., the expression `(RPAQ --)` is ambiguous, because it could define either a variable or a constant. If *ambiguous* is true, you usually want a corresponding entry on `*PFI-FILTERS*` (below).

PFI-PROPERTIES [Variable]

A list used by the default handler for the PUTPROPS form. It associates property names with a type (something more specific than the type PROPERTY) under which objects having this property should be indexed. Each element is of the form (*propname type*). If *type* is NIL or omitted, then objects having this property are ignored. In addition, the default PUTPROPS handler treats all elements of the list MACROPROPS as implying type MACRO.

The initial value of *PFI-PROPERTIES* is

```
((COPYRIGHT)
 (READVICE ADVICE)),
```

meaning that the COPYRIGHT property should be ignored, and the READVICE property implies that the object should be indexed as type ADVICE.

PFI-FILTERS [Variable]

A list describing potential index entries that should be filtered out of the final index. Each element of *PFI-FILTERS* is a list (*type filterfn*), where *type* is one of the types in *PFI-TYPES* and *filterfn* is a function of one argument, an index entry. If *filterfn* returns true, then the index entry is discarded. An index entry is of the form (*name . pagenumbers*). For convenience, an element of *PFI-FILTERS* can also take the form (*type . subtype*), meaning that if an object is already indexed as a *subtype* then it should not also be indexed as a *type*.

The initial value of *PFI-FILTERS* is

```
((VARIABLES . CONSTANTS)),
```

meaning that "variables" that successfully index as constants should not also be listed in the VARIABLES index. This extra pass is needed because the CONSTANTS File Manager command causes expressions of the form (RPAQ *var value*) to be dumped on the file, and at the time this expression is read, it is not known whether there will later on appear a CONSTANTS form for the same variable.

Filter functions may want to call the following function:

(PFI.LOOKUP.NAME *name type*) [Function]

Looks up *name* in the index being built for type *type*. If it finds an entry, it returns it. Index entries are of the form (*name . pagenumbers*). It is permissible for a filter function as a side effect to destructively change another index entry by adding page numbers to it. You might want to do so, for example, in the case where there is a kind of object that dumps two expressions on a file, each of which is a different type (according to *PFI-TYPES*), but you want both occurrences indexed as a single type.

MORE EXPLICIT EXPRESSION HANDLING

The functions and variables described below allow you to completely control how certain expressions in the input file are handled. You can use these hooks to perform custom

prettyprinting, to suppress the printing of some expressions, or to perform indexing more complex than that supported by *PFI-TYPES*.

PFI-HANDLERS [Variable]

An association list specifying explicit "handlers" for expressions that appear on the input file. Each element is a pair (*car-of-form* . *handler*), where *handler* is a function of one argument, an expression read from the file whose first element is *car-of-form*. The handler is completely in charge of indexing the expression and/or printing it to *STANDARD-OUTPUT*. Unless the handler chooses to suppress the printing altogether, it is expected to print at least one blank line first, so that expressions are attractively separated in the listing (see PFI.MAYBE.NEW.PAGE).

PFI-PREVIEWERS [Variable]

This list is used when PRETTYFILEINDEX is used by the SEE command. During the SEE command, real-time performance is important, so it is undesirable to have long delays while reading a very large expression. For example, all the functions in an Interlisp FNS command appear on the file inside a single DEFINEQ expression. If handled in the obvious way, the user would have to wait for the entire expression to be read before any output appeared. A previewer has the opportunity to read the expression in pieces and prettyprint it as it goes.

Each element of *PFI-PREVIEWERS* is a pair (*car-of-form* . *previewer*), where *previewer* is a function of one argument, the *car-of-form*. The previewer is called when PRETTYFILEINDEX encounters an expression of the form "(*car-of-form* " on the file. Its job is to read expressions from *STANDARD-INPUT* (currently positioned after the car of form) until it encounters the closing right parenthesis, which it should consume, and prettyprint the elements appropriately to *STANDARD-OUTPUT*. *PFI-PREVIEWERS* is used only from the SEE command, so indexing is not necessary (but also not harmful, other than to waste some time).

If an expression does not have a previewer, PRETTYFILEINDEX reads the reset of the expression itself and handles it normally, i.e., performs (PFI.HANDLE.EXPR (CONS *car-of-form* (CL:READ-DELIMITED-LIST #\)).

(PFI.DEFAULT.HANDLER *expr*) [Function]

This is the function PRETTYFILEINDEX uses to process expressions that have no explicit handler. It indexes the expression according to *PFI-TYPES* and then prettyprints the expression. You can call this function from your handler if you decide you have an expression you didn't want to handle specially.

(PFI.HANDLE.EXPR *expr*) [Function]

Performs PRETTYFILEINDEX's normal handling of the expression *expr*, including looking on *PFI-HANDLERS*. Handlers and previewers of forms that encapsulate arbitrary expressions, such as DECLARE:, typically call this to process subexpressions.

(PFI.ADD.TO.INDEX *name type/entry*) [Function]

Adds an entry to the index for *type/entry* specifying that *name* occurs on the current page. *type/entry* is either a type or an entry from *PFI-TYPES* from which the type will be extracted.

(PFI.PRETTYPRINT *expr name formflg*) [Function]

Prettyprints *expr*. Optional *name* is the name of the object being printed; if a page crossing occurs in the middle of the prettyprinting, this name will be displayed in the page header. If *formflg* is true, print the expression as code; otherwise as data.

(PFI.MAYBE.NEW.PAGE *expr minlines*) [Function]

Starts a new page if the listing is currently near the bottom of the page and *expr* won't fit, else performs a single (TERPRI). If *minlines* is specified, it is an explicit estimate of how much space the expression will require, in which case *expr* can be NIL; otherwise, the function estimates the size. Handlers should call this *before* calling PFI.ADD.TO.INDEX, so that the page number in the index is correct. The typical handler calls PFI.MAYBE.NEW.PAGE, then PFI.ADD.TO.INDEX, then prints the expression, possibly via PFI.PRETTYPRINT.

OTHER VARIABLES

PFI-INDEX-ORDER [Variable]

A list of types (as in *PFI-TYPES*) in the order in which the various types should appear in the index. Types not in this list are printed in an order of the program's choosing, currently a "best fit" algorithm (print the largest type index that will fit on the page). The initial value is (FUNCTIONS), meaning that the function index will appear first, with no constraints on the order of other types.

PFI-PRINTOPTIONS [Variable]

A plist of print options that PRETTYFILEINDEX appends to the list of print options passed to LISTFILES, thus supplying some printing defaults. The initial value is (REGION (72 54 504 702)), which on standard letter size paper in portrait mode results in left, bottom, top, and right margins of 1", ¾", ½" and ½", respectively. If the print options passed to LISTFILES call for a two-sided listing, the default region is shifted ¼" to the left. If the print options specify LANDSCAPE mode, the default region is ignored. Any REGION option specified in *PFI-PRINTOPTIONS* must be in points; it is scaled appropriately to the actual hardcopy device being used.

PFI-MAX-WASTED-LINES [Variable]

If an expression looks like it won't fit on the current page and there are no more than this many lines remaining on the page, PRETTYFILEINDEX starts a new page before printing the expression. A floating-point value indicates a fraction of the page; an integer indicates an absolute number of lines. The initial value is 12.

PFI-CHARACTER-TRANSLATIONS [Variable]

A list specifying how certain characters should be rendered on the output stream. This is used to get around the poor rendering of certain characters in the default font. Each

element is of the form (*imagetype . charpairs*), where *imagetype* is the type of image stream being printed to and each element of *charpairs* is an alist whose elements are of the form (*sourcecode destcode . looks-plist*), specifying the character code to use on the destination image stream for a specified character code in the input stream. If *looks-plist* is non-NIL, *destcode* is printed in a font obtained by applying FONTCOPY to the current font and *looks-plist*.

The initial value is

```
((INTERPRESS (95 172)
             (96 169 FAMILY CLASSIC)
             (39 185 FAMILY CLASSIC)))
```

meaning if the output stream is an Interpress stream the lister should turn character 95 (underscore) into 172 (left arrow), backquote into left single quote in the Classic font (of the same size and weight), and single quote into right single quote in Classic.

PRINT-PRETTY-FROM-FILES [Variable]

If true, the SEE (in the Exec and Filebrowser), PF and PF* commands attempt to prettyprint to the display, rather than copying the file as it is currently formatted. The initial value is T.

PRINT-PRETTY-BITMAPS [Variable]

If true, then when *PRINT-ARRAY* is true and a bitmap is to be printed to an image stream, the bitmap itself is displayed as an image on the stream, rather than as the machine-readable representation of its bits (of the form #*(16 16)H@@@L...). This variable has no effect on printing to files, such as in MAKEFILE, nor on PRETTYFILEINDEX, which binds it true; thus, changing the value mainly affects the display. The initial value is T.

PFI-DONT-SPAWN [Variable]

If NIL, LISTFILES arranges for a separate process to do the hardcopying (whether using PRETTYFILEINDEX or not) and returns immediately; if T, it makes the listing directly, not returning until it is finished. The initial value is NIL.

LISTING ELSEWHERE THAN THE PRINTER

Ordinarily, you call LISTFILES (or uses the File Browser) to create listings. However, you can also call PRETTYFILEINDEX directly if you want to direct the output elsewhere, such as to an Interpress file:

(PRETTYFILEINDEX *filename printoptions outstream dontindex*) [Function]

Lists *filename*, the name of a Lisp source file or a stream open for input on such a file, printing it and its index to *outstream*. *outstream* is either an open image stream, or NIL, in which case the output goes to (OPENIMAGESTREAM) and the stream is closed afterwards, which results in it being sent to the default printer. If *filename* or *outstream* is open on entry, it is left open on exit. *printoptions* is a plist of options of interest to either LISTFILES or OPENIMAGESTREAM. If *dontindex* is true, no index is produced; this argument is used by the SEE command.

If the file is not a File manager file, PRETTYFILEINDEX takes no action and returns NIL; otherwise, it returns the full file name. However, if *filename* is an open stream, then PRETTYFILEINDEX copies the remainder of the stream to *outstream* (which must be given) using PFCOPYBYTES, and returns the full file name. This is so that the stream does not need to be backed up after discovering that the file is not a File Manager file, an operation not possible for a sequential-access stream.

LIMITATIONS

PRETTYFILEINDEX assumes that the default font, which is used to print the index, is fixed-width.

PRETTYFILEINDEX uses the regular Interlisp prettyprinter. This means that if you have File Manager commands that produce their output in a customized way, e.g., by printing inside the E command, then the output will look different between MAKEFILE and PRETTYFILEINDEX. You can usually remedy this by supplying PRETTYPRINTMACROS for the types of expressions your command dumps (which may also let you replace the E with a simpler P command), or by defining handlers for the expressions (see *PFI-HANDLERS*). PRETTYFILEINDEX already supplies PRETTYPRINTMACROS for most of the customized printing done by the current File Manager: RPAQ, RPAQQ, RPAQ?, ADDTOVAR, PUTPROPS and COURIERPROGRAM.

With the exception of noticing the reader environment and DEFDEFINER expressions, PRETTYFILEINDEX does not interpret the contents of the file. If your file depends on itself for proper prettyprinting or indexing, you need to LOAD (or possibly just LOADFROM) the file first.