Introduction

Welcome to **Pascal Navigator.**

Overview

Reformatting
- Setup
- Block Protection
- File Protection
- Batch Mode
- The Dictionary
- Comments
- Backup

Traversing
- Definition
- Backtrack
- with scope
- Relaxed Mode

Editing
- Cursor Movement
- Block Selection

Semantics vs Syntax

Product Support

Pascal Navigator

Pascal Navigator is an integrated set of tools used in Pascal maintenance:

Reformatting: Extensive source code reformatting capabilities. Now you can read the 'spaghetti' you inherited.

Traversing: A 'point and click' method of finding your way through Pascal source code. This is very valuable with imported programs you are trying to understand.

Editing: A simple editor with Pascal block selection capabilities. For example, you can select a case or if statement with a single keystroke.

Restrictions

Reformatting

To get started, run Pascal Navigator without any command line parameters. Pascal Navigator will go into full interactive mode. When you reformat a Pascal source file, you see the original and new files side-by-side. You may change the options until you get a style you like. You may find some blocks you want to protect from any processing. See Block Protection for how to do this. Pascal Navigator will never change the original file without your permission, and even then will make a backup copy first. After you get a style you like you can start using batch mode. See Batch Mode for details and command line syntax.

Source reformatting is controled by the items under the Control heading in the main menu. These settings stay in effect until changed. The settings are system wide.

Block Protection

See also File Protection

There are times when you want to protect a block of code from any reorganization. This is especially true for tabular layout. Echo mode turns formatting off entirely.

A {+e} inserted as a comment in the source code starts echo mode. From that point on the text is simply copied to the output. It is not even scanned, except to look for a line containing only {-e}, which turns echo mode off. Since echo mode turns off all parsing, echo mode must respect the Pascal block and parenthesis structure. If, for example echo mode includes a begin but not the matching end the indent structure will be incorrect. The {+e} and {-e} comments must appear by themselves on a line. They are not case sensitive.

Do not try to echo part of a statement. The most common place this can cause trouble is where you want to echo part of a type or const definition. The echo block should include the type reserved word and continue up to, but not including the next type, etc. Echo should get all of a declaration or none of it.

Very often the interface to a unit has been carefully hand crafted for readability. It is usually the only part of a unit the end user ever sees. Check **Do not reformat interface part of a unit** box in the General dialog if you want to leave the interface alone.

Block protection affects only reformatting. Protected blocks are treated like any other in traversal.

File Protection

Sometimes you will want to protect an entire file. The {+e} comment on the **first line of a file** tells the reformatter to leave the whole file alone. This is very useful in batch mode with wildcards when some of your files are the way you want them.

Setup

The items under the Control menu do the following:

General	Margins, indentation, etc.
Spaces	Space before and after punctuation
if Statements	if-then-else structure
Indentation	begin-end, case, loop structures
Capitalization Rules	Capitalization rules, dictionary use

Save Style:	Save all options to a file.
Load Style:	Load all options from a file
Reset to Defaults:	Reset all options to default values

General

Scratch File Extension: Pascal Navigator will place the reformatted text in a file with the same name as the source, and the given extension. The file will be in the same directory as the original. The purpose of this file is to give you a chance to look at the result and compile it to make sure everything is OK.

Backup Extension Char: If you replace the original file, Pascal Navigator will make a backup copy of the original in a file with the same name as the source. The extension will be formed by replacing the first character of the source extension with the given character. For example, PROG.PAS will be saved as PROG.~AS. The backup file will be placed in the same directory as the original. If a backup already exists, it will not be replaced unless you say so.

Indent Spaces: indent this many spaces when indent is specified

Line Wrap Indent: indent this many more spaces for continuation lines

Minimum Line Length: stop indenting when lines get this short

Wrap to Punctuation: if multiple lines are needed, attempt to start the continuation lines after the first one of " =" ":" ":=" "(" on the first line. If this is not possible, use the setting in Line Wrap Indent.

Blank Line After Wrap: if continuation lines are needed insert a blank line after the last one

Left Margin: insert this many spaces before each line

Right Margin: last character position used. Note: If a comment or literal (or anything else) will not fit between the left and right margins the line will be written, ignoring the margin settings.

Break Line: insert a highly visible comment line at certain places in the output. The line runs from the left to the right margin. Break lines are placed in the following locations:

(1) before "implementation",
(2) before top level subprograms,
(3) before unit initialization "begin" statements,
(4) before main program "begin" statements.

Break lines are not placed before subprograms nested inside other subprograms. This can help considerably when reading someone else's code.

Filler: fill character used to form a break line. Note: Break lines are deleted during the parse and inserted during the write. A break line is recognized by starting with "(*" and ending with "*)" with nothing but fill characters in between. If you want to change the fill character for a file that already contains break lines, you must first reformat with break lines turned off to remove the old lines, change the fill character, turn break lines back on, and finally reformat again. If there aren't many of them you can delete them by hand.

Pad "function", etc.: add blank characters on the right to make "function",

"procedure", "constructor" and "destructor" all the same length. This is only in effect during an object type definition or the interface part of a unit. This can provide a neater appearance in object type definitions.

Keep Blank Lines: retain blank lines from the original. If this is in effect all of the options to insert blank lines will be ignored.

Keep Tab Characters: keep tab ($09) characters from the original. If this is not checked tab characters will be deleted. Tab characters in literals, comments and protected blocks are always kept. The right margin may be incorrect if this is checked.

UpCase GOTO, EXIT, etc.: shift all "jumps" to upper case. These are GOTO, EXIT, HALT, BREAK and CONTINUE. If this box is checked, it overrides all other case settings for these words.

Blank line before "with": insert a blank line before a "with" statement

Path: Search path used in traverse.

Do not reformat interface part of a unit.: Very often the interface to a unit has been carefully hand crafted for readability. It is usually the only part of a unit the end user ever sees. Check this box if you want to leave the interface alone.

Spaces

These control spacing within a line. A blank character will be inserted in the indicated location.

if Statements

There are many differences of opinion about how an "if" statement should be written. The defaults are for a full "comb" structure. This provides very readable text, and the program structure is obvious at a glance, but many feel that it takes up too much space on the page.

Another common pattern is produced by turning off all options except for Newline Before "else".

Take for example the following:

```
function isAlpha(C: char): boolean;
  begin
    if C in ['A' .. 'Z', 'a' .. 'z']
      then
        isAlpha := true
      else
        isAlpha := false;
  end;
```

or,

```
  if Cp > Length(Line)
    then
      begin
        Block^.Append(Buf);
        Buf := '';
        Next_Line;
      end
    else
      begin
        Buf := Buf + Line[Cp];
        BumpCp;
      end;
```

These code fragments show what the default settings will produce. The "then" and "else" parts remain indented and aligned, no matter how complex the structure.

When you are experimenting with these you may get some unexpected results. In particular if you turn off the Newline after "then" or "else", you probably want to turn off the corresponding indent.

The Blank Line After option inserts a blank line after the entire "if-then-else" structure. If **Keep Blank Lines** is checked in the General menu, the Blank Line Before/After is ignored.

Indentation

This menu controls the layout of blocks. Check the boxes that give you the appearance you like.

Indent: indent the text inside the block

Blank Line Before: insert a blank line before the beginning of the block

Blank Line After: insert a blank line after the block. Note: the blank line is after the block, not after the statement. For example, in a "case" statement the blank line is after the corresponding "end", not after the "case". For a "repeat" statement, the blank line is after the "until" line.

Hanging Indent: This is often called reverse indent. It is the alignment of a line with the next higher nest level. Code is more readable if statement labels are aligned with the outer block. An object definition is easier to read if public and private are offset from the rest of the block.

If **Keep Blank Lines** is checked in the General menu, the Blank Line Before/After is ignored. The Blank Line Before/After option will never produce two blank lines together. For example, if you check both Before and After for case statements, and your program has two case statements together, there will be only one blank line between them.

Indenting subprograms is very useful, especially if the file contains nested (local) functions or procedures. Code readability is vastly improved in these cases.

Declarations are not blocks in the "block structured" sense of the word. These are best explained by example.

```
type
  Switch = string[MaxSwitchLength];
  Operation = (CommandLine, Interactive);

const
  OPMode: Operation = Interactive;

var
  InLines: PLineCollection;
  OutFile: text;
  LineNo: integer;
```

This segment of code was produced using the default settings for declarations. That is, they have Blank Line Before, Blank Line After, and Indent turned on.

Capitalization

Upper: All upper case (example: MY_VARIABLE).

Lower: All lower case (example: my_variable).

Cap: First letter and any letter immediately following an underscore are upper case, others in lower case (example: My_Variable).

As is: Do not change.

Echo first: Capitalization is copied from the first occurrence of a name. This name is not added to the dictionary.

Use Dictionary: Use the dictionary for capitalization. This is how you control names such as "mfOKButton" etc.

Prompt for new words: If you are using the dictionary and Pascal Navigator encounters an identifier not in the dictionary, you will be prompted for the capitalization and the word will be added to the dictionary. If this box is not checked the rules for identifiers will be used for new words, and the new words will not be added to the dictionary. This is always turned off during batch operation.

Reserve Directives: Directives (virtual, forward, etc.) are not reserved words, but many programmers treat them as such. Borland strongly recommends that directives never be used as identifiers. I agree. Check the box if you want to treat directives as reserved words. If you do not check the box they will be treated as identifiers.

Comments

Comments are always difficult in a pretty printer; they follow no rules. Pascal Navigator handles comments in the following way.

If a comment is the first non-blank on a line, it will start a new line in the output. If it is the last non-blank on a line, it will end a line in the output. A single line comment will remain a single line comment. An embedded comment may or may not get wrapped to a new line, depending on spacing, indents, etc. Multiple line comments have the first and last lines handled according to the above rules, while intermediate lines are copied to the output.

If you like to put a comment after each source line, you should echo the blocks where this is done.

The Dictionary

You may want to use table lookup instead of rules for capitalization. For most applications the rules will suffice, but there are cases where you need more control. For example, "ReadLn", "cmCancel", "FileName" and so on follow no particular rules, but you want a standard appearance. This is what the dictionary is for. Each word in the source file is capitalized according to the dictionary.

When a new word is found, you will be prompted for the capitalization if **Prompt for New Words** has been checked. You cannot change the spelling, only the looks of the word. If you enter just a return (enter), the name will be used as is. The dictionary is kept as a text file named PASNAV.CAP in the directory where you placed the .exe files, so you can edit it whenever you like.

The words must be kept in alphabetical order. The order in the dictionary is not case sensitive. If you need to change the looks of a word in the dictionary use any editor to do so, but do not change its location. Do not use a word processor that might introduce extra characters into the file. Use a simple text editor instead. The one you use for source code is OK.

Semantics vs Syntax

Syntax has to do with the proper formation of an expression, while **Sematics** has to do with the expression's meaning.

For example:

X := NoSuch(Y); has correct syntax, but incorrect semantics if X or Y or NoSuch is undefined, or if any of them are of the wrong type.

Pascal Navigator allows you to traverse Pascal code that will not compile. The code need only have correct syntax; it can be semantic nonsense. This is very valuable if you are working with code you got from someone else and you are having trouble understanding it or getting it to work.

Batch Mode

The command line syntax is:

PASNAV [[path]filename.ext [line# [colmn#]] [/brs]]

(Actually, PASNAV_D in DOS, PASNAV_W in Windows.) Example: PASNAV myprog.pas
Example: PASNAV myprog.pas /bs
Example: PASNAV

If the filename is given, Pascal Navigator opens a window containing that file at startup, unless the /b switch is specified. The /b switch indicates batch mode. There must be a space between the filename and the /b switch. These switches are not case sensitive.

line# and column# give the starting position of the cursor. These are mainly used when Pascal Navigator is installed as an IDE tool. For example,

```
  $EDNAME $LINE $COL $SAVE ALL
```

is used in the tool setup for the DOS version. The other switches allowed in batch mode are:

 s Silent: do not show progress during run

 r Replace existing backup file.
    If this is not present, a backup copy will not be made if it would
    overwrite an existing backup.

In batch mode all options are those that were specified during the last interactive session, except that prompting for new words is suppressed. The original source is saved as described elsewhere and a FILENAME.SNP file is not created. All messages are placed in PASNAV.LOG in the Pascal Navigator directory.

Backup

If you replace the original file, Pascal Navigator will make a backup copy of the original in a file with the same name as the source. The extension will be formed by replacing the first character of the source extension with the character specified in the General menu. For example, PROG.PAS will be saved as PROG.~AS. The backup file will be placed in the same directory as the original. If a backup already exists, it will not be replaced unless you say so.

The first time Pascal Navigator is used on a source file, a backup copy is made. An existing backup file will never be replaced without your permission. This is for safety. As you experiment with the options, you want to keep the original intact.

Traversal

Pascal Navigator provides the ability to traverse Pascal source code. The two main commands for traversal are **Definition** and **Backtrack**.

Definition searches the Pascal source code for the definition of the **symbol under the cursor**. The search obeys the Pascal scope rules and uses structure.

Backtrack returns the cursor to the previous location.

Relaxed provides the same function as Definition, but with relaxed search rules.

Inherited displays the name of the inherited object.

Definition

This provides a means to find the location in the source file where an identifier is defined. The identifier definition need not be in the same file as the starting point for the search. The Pascal scope rules are obeyed. If the definition is not found in the current file, the uses list is followed.

The identifier can be a variable, constant, type, label or subprogram.

The search is generally backward in the file, as you would expect. There are several cases where the search is forward:

- -- statement labels
- -- procedures or functions in the interface
- -- procedure or function forward reference
- -- method specifications

Backtrack

Move the cursor back to the location (line, character and file) where the previous Definition was started. Definition/Backtrack can be nested to an implementation defined limit. When that limit is exceeded, the oldest entry is deleted. This is a push through stack.

If the Backtrack stack is not empty, the status line will show the depth with the word **Back** followed by a **<** for each level.

Subprogram

function, procedure, constructor or destructor

Inherited

If you do Definition on the reserved word inherited or use the Inherited command while the cursor is in a method implementation the name of the parent object will be displayed.

With Scope

with statements are not processed, except to note that the cursor is within the scope. **Definition** will not work within the scope; either point to the record name in the with statement, or use Relaxed mode.

Relaxed Mode

The main difference between this and **Definition** is that record and object components are made visible. Since it is common to have the same component name in two different record types**, false hits are likely**. Consider the number of "done" destructors in a typical program.

Relaxed mode will work within a with scope.

Editing

Pascal Navigator provides a simple editor for convenience. It includes line editing as well as block cut, copy, paste and delete. The main strength is the ability to select a Pascal block as a unit.

Line editing:
  Backspace: delete the character before the cursor
  Delete: delete the character under the cursor
  Printable characters: insert at the cursor

  esc: cancel changes to line.
  The changes are not committed until the cursor leaves the line.

  Ctrl-Y: Delete the line.

See Block Selection

Cursor Movement

```
Arrow Keys       -- Move cursor in indicated direction
PageUp           -- Page Up
PageDown         -- Page Down
Ctrl_PageUp      -- Beginning of file
Ctrl_PageDown    -- End of file
Ctrl_LeftArrow   -- Beginning of selection
Ctrl_RightArrow  -- End of selection
```

Block Selection

You can select a block of text in three ways:

Keyboard:
  Ctrl-k,b -- start block
  Ctrl-k,k -- end block

Mouse:
  Drag the cursor over the block **from start to end**.

Pascal Block Select:
  Place the cursor on the block start reserved word, then press Alt-S.

Restrictions

All:
  Source Syntax must be correct. Semantics need not be.
  Conditional compilation is not recognized.
  255 character line length limit.

Traverse:
  Include files are not read.
  With clause not processed.
  The second argument list in a forward reference may not be omitted.
  Relaxed search may produce false hits.
  Type casts are not recognized.

Reformat:
  Right margin not guaranteed if tabs are kept.

Product Support

If you have any questions, suggestions or bug reports, please contact:

James L. Allison
Neptune Systems
1703 Neptune Lane
Houston, TX 77062

Voice 713-488-5722
FAX   713-486-0375

INTERNET: 71565.303@compuserve.com

The **ALL-OR-NONE** blocks are: var, const, label, type, uses, exports

The **SELECTABLE BLOCKS** are: begin, case, if, asm, inline, with, procedure, constructor, destructor, function, for, while, repeat, var, const, label, type, uses, exports