Contents

Subdoc the first and best source code documenting system for Visual Basic.

This is a shareware program which you should register once you've tried it out. The author is determined to deliver a quality product with real benefits. There is no printed manual, but bags of help on-line. Please take the trouble to read the **overview** help screens. The quickest way to get started is to use the sample project after reading the quickstart help screens.

Overview

Subdoc organises VB source code with comments so that code can be reused with ease. You can use Subdoc on the screen while in a programming session or take printouts for reference.

Subdoc does not duplicate any features provided by the VB environment. It does not do nice prints of reams of source code.

There is a very simple mechanism used to make a remark eligible for inclusion in the Subdoc comments. Use two single quotes instead of the standard one. That's all there is to it!

Important:

Subdoc only handles VB (windows) version 2 (or later) source code when saved as text.

Purpose of Subdoc

Subdoc fills the documentation gap for VB2 programmers. Helps you reuse code with the least effort of documentation. VB code is documented in the source so that selected comments are collected against each procedure etc. Subdoc will list all the procedures (with remarks) for a project on the screen or on the printer. Language elements handled as entities by subdoc are:-

Procedures
Types
Declared functions
Constants (project and module level)
Globals (project and module level)
Remarks in declarations section of a module

Example:

```
Substr(stng$,findstr$,replstr$)
Replace occurrences of one string with another.
   ARGUMENTS:-
   STNG$.....String to work on
   Any occurrences of FINDSTR$ will be replaced by REPLSTR$.
   NOTES:-
   *   Circular substitutions ARE allowed.
       eg Call Substr(x$,"HAT","THAT") is OK.
   *   Substitutions ARE NOT recursive. For example if you
       have a string which you want to ensure has no duplicate
       space characters but may have more than two consecutive
       spaces then call more than once.
             eg. x$="123444456"
             call substr(x$,"44","4") ' x$-->"1234456"
             call substr(x$,"44","4") ' x$-->"123456"
```

There is a very simple mechanism used to make a remark eligible for inclusion in the Subdoc comments. Use two single quotes instead of the standard one. That's all there is to it!

```
'' This line will appear somewhere in a Subdoc record
```

There is an additional feature that shows the complete tree of calls made from a procedure and its child procedures and their 'children' etc.

Functions of subdoc described

Selection of source code

One complete project is picked to work on at a time, or alternatively all the source files with the extension "*.SUB" can be treated as a group, thus facilitating documenting libraries of code without reference to a specific project.

Once a selection has been made, Subdoc updates its own records by examining those source files that have been changed since the last time Subdoc was used with this project.

View on screen

The screen is split into two main display areas:

* List of elements
* Current element details

For example the top half of the screen might be showing a list of procedures in alphabetical order, while the bottom half displays the summary remarks for the currently highlighted procedure.

You might have be writing code that calls a procedure. What is the calling convention? Are there any side effects? What does it actually do? How should the return codes be acted upon? List the procedures in Subdoc (that was sitting as an icon all ready for use) and select the named procedure for all the answers in a second.

Printing

Sometimes it is nice to have a hard copy of reference documents.

For a single project (or all *.SUB) the following reports are available:-

Constants (project and module/form level) alphabetically
Globals (project and module/form level) alphabetically
Types alphabetically
Declared functions alphabetically

A report of procedures can be printed for either a complete project or a single module.

A tree diagram can be printed showing calls from a single procedure and subsequent sub-sub calls etc.

Configuration

Printing is much more configurable than the VB facilities! Screen fonts and colours can be changed to suit personal preferences.

Source code requirements

Storage format

VB2 files must be stored in text mode and not binary.
VB1 binary files cannot be used.
VB (DOS) is a totally unknown quantity!

In-code comments

It is not NECESSARY to put double single quotes (DSQ) in the source code for Subdoc to work. However you'll regret having to fathom out what each routine does every time you want to alter it or re-use it.

As a minimum I put a single DSQ line as the first line after the definition of a procedure. eg.

```
Sub fexist%(f$)
'' Return true if file exists
```

This can be extended to give full instructions for using the routine:

```
Sub SetPrinterFont(fcode$)
'' Select either heading font or text font for printer.
'' FCODE$ = H|T (Heading/Text)
'' (No error checking so be careful!)
'' uses ini_ptr_... globals
```

You can still use single quotes for remarks private to a person looking at the insides of the routine, these are normally ignored by Subdoc.

DSQ's are collected in three other cases:

Type definitions DSQ's following a type definition are attached to that type definition record.

Function declarations. DSQ's following `Declare Function` are added to that declaration record.

Modules. DSQ's in the declarations section of a module that do not belong to types or declarations are accumulated as part of the module record. Typically these remarks provide a background to the development, methods, formats and assumptions common to the procedures in the module.

Single quotes are picked up and nicely formatted in the cases of type definition lines, globals and constant definitions. For example Subdoc will record all of the following singly quoted strings:

```
Const MB_OK = 0 ' MSG() ... OK button only
Global Current_month% ' Month number 1...12 (1=Jan)
Dim daysinm%(1 to 12) ' How many days in each month
Type CARD
  Suit as String * 1 ' H|C|D|S=Heart/Clubs/Diamonds/Spades
  Pips as Integer ' Value 1...13
End Type
```

Use of keyword Call

When building a cross-reference table Subdoc needs the keyword `Call` to pick up a call to a 'Sub'. For example:

```
Call Asubprog(1)    ' OK - subdoc will find this
rv% = Afunct%()     ' OK - subdoc will find this
Asubprog 1          ' BAD - subdoc will not treat this as a call
```

Additional requirement

"Terminal" font is required to display calling tree (but not to print).

Using Subdoc

The basic procedure is to pick a project to work with, let subdoc update its records if necessary, then list or print the language elements of interest. Any element can be shown in more detail.

Quickstart will get you up and running in a jiffy.

Quickstart

(The horrible bit)

All the source files for any project being handled by Subdoc MUST be saved in text format. If they are not already then you have to SAVE AS... each one and check text mode. (Change the environment setting while you're about it.)

Sample files SDSAMPLE.MAK/BAS/FRM/SUB are already in this format and if installed on your development directory, will show you what Subdoc is all about.

Start Subdoc

Configure the printer and screen fonts (which takes only a moment). Set the development subdirectory if not \VB.

Pick a project

Pick one of the **projects** from the list. Subdoc has to analyse all the components first time round so this may take a few moments.

Display

Use **View** option to pick the sort of element you want to catalogue, and all items (eg procedures) will be listed in the top part of the screen. The bottom of the screen shows comments associated with this record. (Of course you'll have to put some in before they'll appear here!)

Printing

See what happens when **Print** is picked. It's all quite simple really.

Configuration

PRINTER

Set the margins, fonts and font sizes you prefer for printouts. Depending on the way you format comments, you may find a fixed pitch font such as Courier gives better layout. Generally speaking, a proportional font (eg Arial) will show more text for a given space.

Subdoc will sometimes print at 3/4 the size of the text font. This happens mainly for comments attached to procedures being printed in a calling tree.

If you use punched hole binders you can have the hole positions marked.

You can take a test print. This shows all available fonts point sizes ranging from 6pt to 24pt and also two diamonds. The inner diamond shows the limits of the page you have set with the margin settings. IF ANY OF THE POINTS OF THE INNER DIAMOND ARE MISSING THEN MAKE THE MARGINS LARGER. The outer diamond shows the maximum limits that VB thinks it could work to if allowed (with margins of nothing.)

SCREEN

The little form displayed is a mimic of the main screen divided into three segments with the project list on the left, item list on top and text display at the bottom.

SCREEN LAYOUT

The central status line can be dragged up or down at any time to alter the proportion of the display between the item list and the text display. Resize the whole form to suit the number of items in the item list or show more text.

OTHER

The only option here is to select the base directory for sources. Often this will be \VB. This is where Subdoc creates its working files. Also when asked to collect all *.SUB sources, Subdoc searches the directory tree from here down. You might happily put reusable code in say \VB\UTILCODE and it will be collected into the _ALLSUBS.MAK pseudo project file. Good eh?

Project

Subdoc is rather like VB, it (normally) works on a single project at a time. Picking **Project** from the menu will list the *.MAK files present (On the base directory specified in configuration.)

To allow re-useable code to be catalogued in one pass, subdoc can search the base directory and all lower directories for *.SUB files. Now you can have master library documentation for all the code you might have further use for, in one place.

*If **all subs** is selected then a pseudo project file called _ALLSUBS.MAK is created.*

Project...Load uses any existing .SDP file so that only the minimum amount of work has to be done to update the Subdoc database. However sometimes you may want to use **Project...Reset** ("Reset and load") which erases any pre-existing Subdoc database and starts again from scratch. Use this if you have moved any elements from one module to another.

Remember
All source files to be analysed must be in text format

When a project is selected from the list of projects, Subdoc only needs to analyse those source code files that have been altered since the project was last accessed by Subdoc.

A 'clock' will appear to show progress while Subdoc is working on the sources. *Subdoc is building its own database for the selected project with the extension .SDP.*)

Once all analysis (if any) has been completed then the **View** and **Print** menu bar options will be enabled.

View

Once a project has been selected, the database can be examined with the **View** option on the menu bar. All the sub-options work in the same way. (Although **Calls** may take additional time to build a cross-reference table.) The index of items corresponding to the type selected is shown in the top part of the display. (The form can be resized or the central status line dragged up or down to optimise the display.)

There are two **Search** options:

by element name is a wildcard search using the same parameters as the VB operator `like`. A * is automatically added to the end of the string. For example"Load" will match LoadFile% (), LoadList&, LoadedFlag%.

by remark text is a search of the definition line of the element and any remarks associated with that element. *Wild cards are not allowed, but the search is always insensitive to case.*

Picking one of the items in the list will display details in the lower part of the display. (The form can be resized or the central status bar dragged up or down to optimise the display.)

Even if there are no remarks to display, the status line will report some information about the selected item.

(To print the same information as shown on the lower part of the display select **Print** then **Current Display**.)

Calls can take longer to list because if one module in a project is changed then the whole project has to be cross-referenced. The calls display can take a little longer, as all the calling avenues for a particular procedure are explored. **Only those procedures with one or more calls from them are shown in the list**.

Print

The simplest way to see what these options do is to try them. SDSAMPLE can be used as an example if installed on your development directory.

Procedures...

Procedures can be printed for a single module or the complete project. Function return types are determined by explicit declaration or by the DEFtype in force at the time.

To save space on the printout (and so get in long lines of function arguments) some of the `AS...` clauses are replaced by type characters and `BYVAL` is omitted. User-defined types are retained.

Example source

```
Function fexist(f as string) as integer
'' Return true if file F$ exists
'' (Just a DIR$() really)
```

becomes:

```
Integer <------ FEXIST%(f$)
        Return true if file F$ exists
        (Just a DIR$() really)
```

DSQ's can be placed anywhere in the body of the procedure to be collected by Subdoc, but I can't find any reason for not putting them right at the top.

Declarations

All declarations for the project are printed in alphabetical order. As with procedures, the format is modified from correct syntax to give a more compact printout. DSQ's to be attached to a function declaration must follow the declaration line.

Types

All the user-defined types for the project are printed alphabetically. Each entry shows the type of the element, the element name and any remark associated with that element. Additionally, DSQ lines following immediately in the source after the `End Type` statement are included in the entry. This gives scope for a general description of the type. Single quotes on the same line as an element of a type are included in the type record.

Example source

```
Type CARD
  Suit as String * 1 ' H,C,D,S=Heart/Clubs/Diamonds/Spades
  Pips as Integer ' Value 1...13
End Type
''      Used in the Pack() for representing playing cards
```

becomes:

```
  Suit     String *1   H,C,D,S=Hearts/Clubs/Diamonds/Spades
  Pips     Integer     Value 1...13
  Used in the Pack() for representing playing cards
```

Modules

All modules in the project are listed alphabetically. Any DSQ's in the declarations section (not otherwise attached to types or declares) are printed. Typically this facility is used to give an overview of the module's function, development history, ownership, background information of a general nature.

Global variables

This prints all the variables with project level or module level scope. The printout is organised alphabetically with project and module globals mingled. Variables with project scope (`Global...`) are

shown in bold while those at module level (`Dim...` in declarations section) are in plain type.

Singly quoted remarks on the same line as the Global/Dim are shown. For example, source in APROG.BAS

```
Global NoHoles% ' Number of punching holes ....
```

becomes

APROG.BAS NoHoles% Number of punching holes. The allowed values are 0,2,3,4

Constants

This prints all the constants with project level or module level scope. The printout is organised alphabetically with project and module constants mingled. Constants with project scope are shown in bold type while those declared at the module level are in plain type.

The value of the constant is shown. Singly quoted remarks on the same line as the definition are printed.

Example source in APROG.BAS

```
Const DF_SRCCOPY& = &HCC0020 ' Raster OP code (copy pen)
```

becomes:

APROG.BAS DF_SRCCOPY& &HCC0020 Raster OP code (Copy pen)

Current display

This option is only enabled if there is a display in the lower part of the screen. This is the only method of printing a calling tree. The quality of the printout is not as good as the dedicated printouts described above.

Printing calling tree

The calling tree illustrates the calls that a particular procedure makes. A printout is only ever made for one top-level procedure at a time.

Two options are provided when printing, the printed report being a development of the bare-bones tree shown in the display.

[x] Print remarks

Check if you want the DSQ remarks for each procedure to be printed the first time a procedure appears in the tree.

[x] Expand repeats

Check only if you want repeated sub-trees to be expanded. If this is left unchecked when a repeated sub-tree is encountered it is indicated with ellipsis (...) after the top procedure name.

Note: The layout of the printed reports might seem a little strange at first, however they are designed to be bound into a ring binder or perhaps stapled together along the left edge. Now the key words are easy to scan as you flick through the pages.

About the author

I'm a freelance programmer.

When I'm not happy with the tools at my disposal I evolve my own. Subdoc is a development of an earlier QuickBasic program and coding standard that has saved me hours of fumbling through my library of QB source code. VB1 was no help because source code couldn't be saved (easily) in ASCII for cataloguing. With VB2 we can now get down to the serious business of documenting code for ease of re-use.

When I have a need I create the necessary technology to satisfy it. For example, to bring Subdoc to you with the sharp edges removed I've rehashed my RAT routines, re-implemented quicksorts for speed, and spent hours designing an in-progress indicator.

If you are part of a software manufacturer and seller who is happy to invest 20 man-years in a mega-business product that'll make spreadsheets look like chickenfeed then contact me.

If you want to use TWAIN protocols with VB then contact me. If you need a Clarion programmer then contact me. If you want to do things with HPCL5 then contact me.

Finally, please register. Let me know if the thing is useful or otherwise and then the next version might incorporate your suggestions.

Trouble

Check

Are all sources in ASCII?

Do all Calls explicitly use the CALL keyword.

Problem: The footer appears at the top of each page instead of the bottom. Solution: Increase the bottom margin.

If all else fails then send me a description of the problem.

Files used by Subdoc

Everyday use

*.SDP Database of collected entities. 1 per project.
*.XTB Cross reference table. 1 per project.
*.XPN Cross reference labels. 1 per project.
_ALLSUBS.MAK Pseudo-project file for all SUB's

System files

\WINDOWS\SUBDOC.INI	configuration file.
SUBDOC.EXE	This executable
SUBDOC.HLP	This help file
VBRUN200.DLL	Run-time library.(Not supplied)
\WINDOWS\SYSTEM\CMDIALOG.VBX	Custom control DLL

Various temporary files are created with the extension .TMP

Getting started

SDSAMPLE.*	Example application

Installation

SDOC210.EXE	Self extracting ZIP containing...
INSTALL.TXT	Installation instructions
SUBDOC.EXE	Program
SUBDOC.HLP	Help file
CMDIALOG.VBX	Custom control DLL
SDSAMPLE.*	Sample application files

Apologies for putting more files into WINDOWS but this is the only place I can be dead sure of finding them.

Version information

THIS IS VERSION 2.10

Version 1.00 was first release. March 1993
Version 1.20 was a bug fix. April 1993
Version 2.00 intended to be the real thing, June 1993.
Version 2.02 fixes odd bugs and allows spanning discs. July 1993.
Version 2.10 allows much larger projects to be cross-referenced. February 1994.

Keep sending me feedback because this is really useful, and it gives me no excuse for letting anybody down.

IMPORTANT NOTES : Thoughts for later versions:

* Calling tree in source code order
* Highlight events that trigger calls
* Quick source code print

and

* Your suggestions

Tech talk

Techie shall talk unto techie. Take your pick from these snippets. **These sidelines have no relevance to the operation of Subdoc**. However there may be a nugget in this section that might bust a log-jam, or trigger a line of thought, or simply appeal as a good idea.

Source code for sale

RAT (Random Access Text) routines in VB2 source code format.
UK pounds 45.00 (+VAT 17.5% in UK and EC)
Money back if not satisfied
Fuller description available via
Compuserve E-mail or
Letter mail (send self addressed envelope).

Sorting

Speed is the number 1 priority with sorting. Strings are slower to process than numbers and goodness knows how VB handles variants internally. In QuickBasic there is the SWAP command which when used with strings swaps pointers, but in VB we need to allocate and manipulate pointers ourselves...
....Can be done.

If your sorts are slow try
a) Better algorithms
b) Using pointers
c) Or get hold of sources for sorts from somewhere

Or

d) How about creating a (sorted) listbox which never need be displayed then pumping all your keys into it (with pointers if needed as `.ItemData`). Voilla!

Hint: Try d) first.

Random Access Text

In QB I have had a set of routines that allowed me to store arrays of text in a file, encrypted and indexed by a 'module number'. This proved to be useful and versatile. Now I could have used this system in VB for Subdoc. (You may appreciate that Subdoc collects chunks of text and stores them indexed by element procedure and module. We may have one or 100 lines of text of any width.) Unfortunately the QB routines were rather inefficient when large amounts of updating was to be carried out, so improved technology was needed. My RAT routines are the result. They store text efficiently, re-use deleted portions of the file, and provide a flexible indexing system. For the average programmer there is just one high-high level call that does everything.

One file can be used for a variety of record types since any chunk of text can be identified by various attributes in the index.

Could be used as the storage mechanism for the documentation of a network. Nodes, links, hardware components, software, licences and maintenance records could all use the same file with the benefit of great flexibility in the information stored in each record. In the UK all personal information has to be secure, the RAT routines provide encryption.

For sale

SDSAMPLE - Example project

A working demonstration program. Illustrates how form.width and form.height are no good for internal dimensions but can be modified. (How to be clever and stupid at the same time - guess who gave up on `.Scalemode` at an early stage and did it the hard way.)

Note: ScaleWidth and ScaleHeight are the things to use whenever you want to work on an object's surface. Width and Height give the external dimensions. This is not made clear in the documentation!

Thus:

```
Printer.scalemode=1             ' twips
wtwips% = printer.width         ' how wide is printer
swtwips% = printer.scalewidth   ' width we can use
If wtwips% <> swtwips% then     ' ought to be same
    Print "MORAL:Never trust .Width" ' but aint!
end if
```

In-progress indicator

Don't you just hate the status bar that creeps from the left, bang wallop in the centre of the screen? I've been looking for something a bit more interesting.

* I tried pictures of flowers growing from shoot to bloom. Pretty, but bitmaps doubled size of executable!

* I tried water going down a plug hole. Nearly made it but it lacked a little realism even with lots of details.

* Settled on the simple 'clock'.

What ideas can you suggest?

* What about cartoons which use mainly drawing rather than bitmaps? A teddy bear parachuting from an aeroplane? An ice cube melting? A sundial? .....

* What about activities for the frustrated user? Anagrams, quizes, a things-to-do list, doodle pad, unusual words defined, mnemonic dictionary, 1001 Jokes,.....

Currently this research has no home, and no prizes are on offer so:

1) Anyone want to sponsor a collection?
2) Try out a few ideas in rough and E-mail me.
3) I have a suggested spec.

The first person to submit a creeping company logo will get a nasty dose of the chinese-chip-rot right where it hurts!

Suggested specification for in-progress indicator

```
Integer <---- INPROGFUNC(SoFar&,OutOf&,Pic as pictureBox)
" Display in-progress indicator
" SOFAR& is progress so far made (0 to OUTOF&)
" OUTOF& is maximum value expected of SOFAR&
" PIC is control on which the images are drawn
" Return
" 0 ... (Reserved for) OK completion
" -1 ... (Reserved for) Problem
" -2 ... (Reserved for) Interrupted by watchdog timer
" Other +ve values: milli seconds used during call
" Other -ve values: whatever use you wish
" OUTOF& special values
" 0 ... (Reserved for) Finished with display
```

If this spec won't fit your needs then use another, but remember that you have `Pic.Tag` that can be used for passing string arguments, and various states such as `Pic.visible=False` can be used to indicate that the function should re-initialise.

Global memory

It only takes a few calls to the API and you can happily use memory from outside VB and talk to other applications using it. Knowing how to do this can be important if you are interfacing with DLLs that return handles to global memory then expect you to de-allocate it (or vice versa).

All you need to know is:

Globalalloc reserves a specified amount of memory and returns a HANDLE to that memory. (The actual physical location may move or even be shunted out to disc, as Windows manages memory.)

To freeze a wandering memory block reserved with **Globalalloc** and obtain an actual ADDRESS so we can use it, we have to call the **GlobalLock** function (which returns the first address location of the block.)

After completion of the immediate operation we MUST MUST MUST call **Globalunlock** which allows Windows to control the memory block again. The address we obtained earlier is now invalid and must NEVER NEVER NEVER be used.

_hread gets a specified number of bytes from a file and puts them into a buffer starting at a specified memory ADDRESS. **_hwrite** is the twin of **_hread**.

Example

The outline of the first function shown in the example is:

How much memory will we need?
Reserve it, and get a handle to this reserved block.
Fix it, get the current actual address of the start of block.
Transfer bytes from file to block.
Free the block (address no longer valid)

The second function can get to this data because we pass the handle to it. When it is ready it uses Globallock to get the actual address then as soon as possible unlocks it.

If this is just what you need then you ought to read Petzold, or the API files with VB Professional for safety's sake. However you can go a long way by looking at the example. It is not ready to run, but the code has been extracted from a working application.

Programming Windows 3.1

Often known as **Petzold.**

This is the standard book on the subject. You need a rough working knowledge of C programming to understand it. Clear and interesting. Essential, if rather gruesome, reading for C programmers. Lucky VB programmers have all the Windows chores done for them so only need this if playing games with the API. (This is not a book about VB.)

Programming Windows 3.1
3rd Edition
Microsoft Press,Washington 1992
Charles Petzold
ISBN 1556153953

API function declarations

Note: _hread is new to Windows 3.1

```
Declare Function globalalloc Lib "Kernel" (ByVal wflags As Integer, ByVal
dwBytes As Long) As Integer
Declare Function stretchdibits Lib "GDI" (ByVal hDC%, ByVal X%, ByVal Y%,
ByVal Dx%, ByVal Dy%, ByVal SrcX%, ByVal SrcY%,...
 ...ByVal wSrcWidth%, ByVal wSrcHeight%, ByVal lpvBits&, lpBitsInfo As
bitmapinfo, ByVal wUsage%, ByVal dwRop&) As Integer
Declare Function Globalfree Lib "Kernel" (ByVal hmem As Integer) As Integer
Declare Function globallock Lib "Kernel" (ByVal hmem As Integer) As Long
Declare Function hread Lib "Kernel" Alias "_hread" (ByVal hFile As Integer,
ByVal lpBuffer&, ByVal dwbytes As Long) As Long
Declare Function globalunlock Lib "Kernel" (ByVal hmem As Integer) As Integer
```

Function code.

```
Function FetchBitMap% (f$, Startbyte&, bmi As bitmapinfo)
''  F$ is the full name of a monochrome bitmap which will be loaded into
memory
''  StartByte& is the position of the first byte of the BITMAPFILEHEADER
structure.
''      (For a file that is just a bitmap then this will be 1.)
''  BMI is a blank BITMAPINFO structure which will be filled
''  Returns handle to global memory that identifies location of bitmap bits
''  returns 0 if failed

If Not fexist%(f$) Then                  ' catch silly
    FetchBitMap% = 0
    Exit Function
End If
If Startbyte& < 1 Then Startbyte& = 1       ' catch silly

'        Find out how large the bitmap is
'        --------------------------------
ff = FreeFile
Open f$ For Binary As ff
Dim bmf As BITMAPFILEHEADER
If Startbyte& < 1 Then Startbyte& = 1       ' catch silly
Get #ff, Startbyte&, bmf                 ' read file header
Get #ff, , bmi                           ' read info header
'   ***** shortcuts and assumptions made here! *****
FirstFileByte& = bmf.bfoffbits + Startbyte&    ' actual first byte in file of
mono bitmap
Seek #ff, FirstFileByte&                 ' point to start of data

nofilebytes& = bmi.bmiheader.bisizeimage    ' no of bytes of data to read in
If nofilebytes& < 1 Then                      ' trap other peoples errors!
    nofilebytes& = bmf.bfsize - bmf.bfoffbits
End If

'   allocate memory and get address handle
wflags% = 2                                    ' moveable
hmem% = globalalloc(wflags%, nofilebytes&)     ' allocate and get handle
If hmem% = 0 Then                              ' deal with problem
```

```
    Close ff
    FetchBitMap% = 0
    Exit Function
End If

fhDOS% = FileAttr(ff, 2)          ' get dos file handle
amem& = globallock(hmem%)  ' get memory actual address

    'NOTE: should trap error value from global lock here
    nobread& = hread(fhdos%, amem&, nofilebytes&)   ' huge read from file to
    mem block
r% = globalunlock(hmem%)   ' release global lock ASAP
Close ff                                        ' done with file
FetchBitMap% = hmem%              ' return handle
End Function


Function fillpicture (pic As PictureBox, sourceextent As extent, bmi As
bitmapinfo, hmem%)
''  This will stretchblt from the bits already in memory as identified by
''  the handle to global memory Hmem% and the bitmapinfo structure BMI.
''  The destination rectangle is always the full rectangle of the PIC picture
control.
''  The source rectangle is identified by SOURCEEXTENT.
''  NB inputs arguments for sourceextent assume origin is TOP LEFT.
''  If any values of SOURCEEXTENT are out of range then they will be clipped
and these
''  alterations returned with the structure.
''  RETURN  number of lines written to picture (0=fail)
Const srccopy = &HCC0020
Const fucoloruse = 0
'   The source bitmap origin is bottom left so invert vertical values
syt% = bmi.bmiheader.biheight - (sourceextent.top + sourceextent.height)
If syt% < 0 Then syt% = 0

amem& = globallock(hmem%)  ' fetch actual memory address
fp% = stretchdibits(pic.HDC, 0, 0, pic.width, pic.height, sourceextent.left,
syt%, sourceextent.width, sourceextent.height, amem&, bmi, fucoloruse,
srccopy)

r% = globalunlock(hmem%)   ' let windows manage memory
fillpicture=fp%                   ' return value
End Function
```

Saving code in text format

To set the default format that VB uses to save modules

Select **Options** from the VB menu bar.
Select **Environment**.
If **Default Save As Format** is not set to TEXT then change it.
THIS WILL ONLY BE EFFECTIVE ON MODULES CREATED THEN SAVED SUBSEQUENTLY.

To save an existing module in text format.

Load the project into VB.
For each BAS/FRM/SUB file in project:
    Highlight in list
    Pick **File** - **Save As...**
    Check the **Save as Text** box
    Click **OK**
Once a module's format has been set in this way it retains the new format unless changed again by this procedure. Therefore you only ever have to do this once for each module.

Calling tree

The calling tree shows one branch for each procedure that is called by a selected procedure. These branches may themselves have branches and so on.

Only one branch is added even if one procedure calls another many times.

Branches at the same level are listed alphabetically - not in order of execution.

Warning:

Control can pass between procedures in other ways than explicit calls. Events can be triggered in all sorts of ways. For example `TextBox.Text = "Something else"` will trigger `TextBox_Change()`. These events can be quite important (and need careful watching) in VB. (If your code is designed to work like this I suggest you put a warning in the comments to the effect that certain other procedures service this one. - *A later version may have a way of highlighting or documenting these triggers*.)

To print the calling tree for a specific procedure:

View - **Calls**
then
Print - **Current display**

Only procedures with 'children' are shown in the item list.

Calls to declared functions are not shown.

Contacting the author

Author:
Peter Fox
Mail:
2 Tees Close
Witham, Essex
CM8 1LG
ENGLAND
E-Mail:
Compuserve 100116,1031
Internet Peter@pfox.demon.co.uk
Telephone:
0376 517206 (UK)
+44 376 517 206 (International)

Please, Please, PLEASE don't use the phone at midnight UK time! If you leave a message on the answering machine I probably won't reply anyway. Use E-Mail or paper mail if you want a reply.

IMPORTANT: I get lots of mail that I can't handle for two simple reasons:
1 No name or address or indecipherable.
2 Rambling, never seems to get to a definite point.

Use E-Mail if at all possible and save everyone hassle eh?

Double-single quotes

In VB a single quote is a normal remark or comment.
In Subdoc, two of these together are a signal that the remark is to be collected as part of the documentation.

```
' an everyday remark
'' to be collected by subdoc
```

Modules

In VB the term 'Module' is used to describe source without a form. In Subdoc 'Module' is any source code file including .FRM files.

.VBX files are NOT modules.

Subdoc will collect source code files with almost any name if they are specified as such in the .MAK file. To build libraries of source code that Subdoc can gather into a single 'project' for documentation purposes the reusable code modules should have the extension .SUB. I use this convention because it means I can put code that I think may come in handy in another project into a .SUB and maybe make a little extra effort to make that code general purpose. Having done that, unless I make a note of that asset (using Subdoc to maintain my library catalogue) then the investment has been wasted.

Language elements

Procedures

These are identified by

```
Sub ...
```

or

```
Function ...
```

The keywords `Static` and `Private` are allowed to precede `Sub` or `Function`.

Declarations

Declarations specify calls made to code outside VB. Typically API calls.

These are always located in the declarations section of a module and start with the keyword

```
Declare ...
```

Deftype

Subdoc uses `Deftype` statements to determine what sort of value a procedure returns if it is not explicitly specified. Thus:-

```
DefInt A-Z
   ...
Function ThisReturnsInt(Aval)
```

Important note:
Subdoc does not evaluate the type of procedure arguments. So in the above example no type is associated with Aval. It is good practice to be explicit with type characters or `AS...` clauses.

Globals

Subdoc identifies variables with project level scope by the keyword `Global...` in a definition section of a module. Module level scope is identified by the keyword `Dim...` in a definition section of a module.

Quick definition: A 'Global' in Subdoc is a variable with more than procedure level scope.

Constants

Subdoc identifies constants with project level scope by the keywords `Global Const...` in a definition section of a module. Module level scope is identified by the keyword `Const...` in a definition section of a module.

Quick definition: A constant in Subdoc is any constant with more than procedure level scope.

Calls

Calls from one procedure to another in the project are identified by:

a The keyword `Call ...`

b Use of a function described in the project.

Note: The syntax to call a sub-program with no return arguments MUST include the keyword `Call` for Subdoc to pick it up in the cross reference table. For example:

```
Call DoSomething(Avalue%)                    ' Good
Dosomething Avalue%                          ' Bad
```

User-Defined types

Subdoc identifies a user-defined type with the keywords `Type` at the start of the type definition and `End Type` at the end.
Type definitions always appear in the declarations section of a module.

Shareware

The shareware concept is simple:-

Try it,
then
buy it
or
bin it.

In any case feel free to pass it on to a pal.

Unfortunately this isn't free software. If you like Subdoc and want to use it then you must become a registered user.

The benefits of registration are:-

Technical support via E-Mail.
Technical support via post.
Latest version. Check release plans
Legality.

Registration fee:

UK Sterling 17.00 or
US$ 35.00

How to become a registered Subdoc user

Please register. Registration is not expensive. By registering you are supporting quality software development, and it is your chance to say what you'd like to see in later versions.

The benefits of registration are:-

Technical support via E-Mail.
Technical support via post.
Latest version please check release dates
Certificate of registration
Legality.

Notes:

* There is no printed manual.
* Source code is not available.
* I have done my best to see that the program works and will do my best to fix problems that may arise. However, in the last resort, you might have to be satisfied with your money back (as your only remedy) if I cannot fix an identified problem.
* There may be alternative payment arrangements that have been put in place since this version has been released.

Registration is simple!

(The registration screen will become a **Print** menu option once a registration form has been printed.)

Credits

Written entirely in Microsoft Visual Basic version 3 for Windows.

Thanks to Dave Baldwin for being able to use his RTFGEN program to assist in building this help file.

Thanks to Rudolf Haupt and many others for their feedback and patience with earlier versions.

Thanks to Jose Luis Lozano for reporting the limits on cross referencing large projects in version 2.0. As a result 2.1 was created.

TWAIN

This is a protocol and API for acquiring raster image data. Typically, but not exclusively, from scanners. Supported by major companies such as Aldus, Caere, Hewlett-Packard, Logitech and Kodak. The object is for applications programmers to write a single interface that will run with a number of different data acquisition devices. Try one of the above companies for more info.

Clarion

An excellent DOS database applications builder from Clarion Software Corporation.

WELCOME TO SUBDOC

Getting started should be quite simple. Check that you have followed the installation instructions in the file INSTALL.TXT that was Zipped with this program.

This is a shareware program. If you like it and use it then you MUST register it. In any event you can pass it on to your pals so long as you don't charge for doing so.

I have done my best to see that the program works, and will do my best to fix problems that may arise. However, in the last resort, you might have to be satisfied with your money back (as your only remedy) if I cannot fix an identified problem.

F1 help should be available at all times. Please browse this help now. Topics you may find useful are Contents and Quickstart

Peter Fox 4th June 1993

Ini file

All the parameters in the **Screen** and **Printer** sections are controlled by the program and should never need manual alteration.

The parameters in **Various** can be changed using NOTEBOOK (for example).

To suppress the confirmation on exit box, replace Y with N.

On registration you will be told how to edit the INI file to remove the copyright message that appears on every printed page and replace it with your own footer.

What you get by registering

You have tried Subdoc, liked it, now want to do the decent thing and register it. Good! Here is what you are getting for your registration fee.

UK sterling 17.00 or US$ 35.00

1 The legal continued use of this version of Subdoc.
2 A certificate of registration.
3 Support from the author via E-mail.
4 Either immediate delivery of current version if this is later than your current version, OR entitlement to shipment of the next version of the program when it arrives.

(No manuals or source code will be supplied. The software that you have now is the real thing so you don't need anything else to get productive.)

If you do not receive a registration certificate within say two weeks of registering then please contact the author direct for an explanation!

To register a site licence (double charge) simply register two copies.

How to register

Your piggy bank has been raided, but sending cash by post around the world is not the best of ideas, so how can you register more easily?

1 Cheque drawn on UK bank.
Please fill in the form that is printed when picking this option and send to the author.

2 Via Compuserve.
If you use Compuserve then you can register in a few moments. The very simple instructions are printed when this option is selected.

3 Payment via shareware house.
Many shareware houses provide a registration service. You register through them using credit card or other payment methods. However you should not assume this is the case, either call your favorite source of shareware and enquire, or consult the text printed if this option button is clicked.