October, 2001

Advisor Answers

Search through a whole project

VFP 7.0/6.0

Q: I am often interested in searching all the source code in a VFP project for a certain text string (a file or procedure name, for example). I am not sure how to do this. I know how to search the source of a particular module, but I'm not sure how I can search every module in the project.

–Scott Mabey (via Advisor.COM)

A: There's no built-in way to search for a string in an entire VFP project. You can search in a particular file using the Edit-Find menu item (or its corresponding shortcut, Ctrl+F). When you search in a form or class, choosing All objects as the Scope of the search ensures that you search in every method. (One caveat: when more than one method editing window is open, a search omits the methods shown in editing windows other than the one that started the search.)

In VFP 7, the Find dialog has been enhanced to allow wildcard searching. (See the "Find Dialog Box" topic in Help for details.) VFP 7 also allows you to search without bringing up the Find dialog. Highlight some text and press Ctrl+F3 to search for the next occurrence of the highlighted text. Ctrl+Shift+F3 searches backwards for the highlighted text.

The Filer utility provided with VFP (in the Tools\Filer directory) lets you search for files with a specified name or containing up to three specified text strings in a directory and its subdirectories. But the Edit button in Filer brings up the file as text, not in its native tool. In addition, projects don't require all files to be in a single directory tree.

Fortunately, VFP's open architecture lets us build our own project search tools. In VFP 6 and later, a project is accessible as a COM object, which makes it easy to access every file in the project. In earlier versions, you can search through the project table, since a .PJX file is just a .DBF with a special extension. The actual files that contain code are either text files (.PRG, .H) or are tables with special extensions (.VCX, .SCX, .MNX). VFP has excellent string handling

capabilities, so once we get our hands on a piece of code, it's easy to see whether it contains the search string.

I put all this together and wrote a rudimentary project search tool for VFP 6 and later. It's a class called cusProjectSearch, based on the Custom class (and is available on this month's Professional Resource CD). The class has four custom properties:

- aMatches – an array containing the matches from the most recent search
- cSearchString – the string to search for
- nMatchCount – the number of matches in the most recent search
- oProject – an object reference to the project to search

It has quite a few custom methods, but the key method is Search. You call Search, optionally passing a search string and a project. You can pass the project using an object reference or its name. Here's the key code for the Search method. A lot of error-handling code is omitted here for space reasons (but is included in the version of the class on the PRD):

```
LPARAMETERS cSearchString, uProject
  * cSearchString = the string to search for. (Optional)
  * uProject = project to search. Can be either
  *            object reference, in which case the
  *            project must already be open, or
  *            filename with path, in which case the
  *            project is opened, then closed. (Optional)

LOCAL lWasOpen

* Check parameters and set up search
DO CASE
CASE VARTYPE(m.cSearchString) = "C"
   This.csearchstring = m.cSearchString
ENDCASE

DO CASE
CASE VARTYPE(m.uProject) = "O"
   * Check object type, then proceed
   IF UPPER(m.uProject.BaseClass) = "PROJECT"
      This.oProject = m.uProject
      lWasOpen = .T.
   ENDIF

CASE VARTYPE(m.uProject) = "C"
   * Is the project already open?
   IF TYPE("_VFP.Projects['" + ;
      FORCEEXT(JUSTFNAME(m.uProject),"PJX") + "']") = "O"
      This.oProject = _VFP.Projects[ ;
         FORCEEXT(JUSTFNAME(m.uProject),"PJX") ]
```

```
      lWasOpen = .T.
   ELSE
      * Make sure file exists. If so, open it.
      IF FILE(m.uProject)
         MODIFY PROJECT (m.uProject) NOSHOW NOWAIT
         This.oproject = _VFP.ActiveProject
         lWasOpen = .F.
   ENDIF

ENDCASE

* If we get this far, we have a valid string and project.
* Now go through all files and check each one.
LOCAL oFile

This.nMatchcount = 0
FOR EACH oFile IN This.oproject.Files
   This.SearchFile( oFile )
ENDFOR

IF NOT lWasOpen
   This.oproject.Close()
ENDIF

RETURN This.nmatchcount
```

This code checks the parameters, and if they pass muster, stores them to the appropriate properties. If the parameters are omitted, but the properties are already set, that's acceptable. If anything goes wrong in this process, an appropriate error is fired, so that the error handler can deal with it. (However, this simple class doesn't actually have an error handler, so VFP's default error handler gets called.)

If all the tests pass, the method loops through the files in the project and calls the SearchFile method for each. Finally, if the project was opened in this method, it's closed. The method returns the number of matches found.

The SearchFile method has a simple role. It determines the type of each file and calls an appropriate method to process it. Here's the code:

```
* Search a single file for the search string.
LPARAMETERS oFile

WITH oFile
   * Find out what kind of file we have
   DO CASE
   CASE INLIST(.Type, "P", "Q", "T")
      This.SearchProgram( oFile)
   CASE INLIST(.Type, "K", "V")
      This.SearchMethods( oFile)
```

```
   CASE INLIST(.Type, "M")
      This.SearchMenu( oFile)
   CASE INLIST(.Type, "d")
      This.SearchStoredProcs( oFile )
   CASE INLIST(.Type, "R", "B")
      This.SearchOutput( oFile )
   OTHERWISE
      * Nothing to search
   ENDCASE
ENDWITH

RETURN
```

The various SearchX methods dig into the files. They're all fairly similar. Here's the key processing loop for SearchMethods. Before it gets to this point, the method opens the VCX as a table with an alias of __SearchFile.

```
* Check each record for code
SCAN
   IF NOT EMPTY(Methods)
      This.SearchField( Methods, .T., oFile.Name, ;
                        ObjName, .T.)
   ENDIF
ENDSCAN
```

SearchMethods opens the file as a table and looks at the Methods field of each record. If it contains anything, that code is passed to the SearchField method, which is the real workhorse of the class:

```
* Search for the specified string in a single
* field of one of the "X" files (VCX, SCX, FRX, ...)
LPARAMETERS cContents, lGetProcName, cFileName, ;
            cObjName, lDropNameLine
   * cContents = Field contents to be searched
   * lGetProcName = Should we search for the name of the
   *                procedure containing the search
   *                string?
   * cFileName = Name of the file name containing
   *             cContents
   * cObjName = Name of the particular object containing
   *            cContents
   * lDropNameLine = Does cContents contain method code,
   *                 where the "PROCEDURE" line is only
   *                 implied? If so, line numbers must
   *                 be adjusted to omit that line.

LOCAL nMatchPos, nMatchCount, aCodeByLines[1]
LOCAL nReturnCount, nProcLine, nCodeLine, cCodeLine
LOCAL cProcName, nParmsBegin, cFirstWord

* Break code into lines
ALINES( aCodeByLines, cContents )
```

```
* Find first match
nMatchCount = 0
nMatchPos = AT( This.cSearchString, cContents )

DO WHILE nMatchPos <> 0
   nMatchCount = nMatchCount + 1
   * Figure out which line contains the match
   nReturnCount = OCCURS( CHR(13), ;
      LEFT(cContents, nMatchPos - 1))
   nCodeLine = nReturnCount + 1

   IF lGetProcName
      * Search backward for procedure name
      * starting from current line
      nProcLine = nCodeLine
      * Get rid of tabs and leading spaces
      cCodeLine = LTRIM(CHRTRAN( ;
                  aCodeByLines[nProcLine], CHR(9), " " ))

      nHeaderEnds = AT(" ", cCodeLine)
      cFirstWord = UPPER(LEFT( cCodeLine, ;
         IIF(nHeaderEnds=0, LEN(cCodeLine), ;
         nHeaderEnds -1 )))
      DO WHILE ("PROCEDURE" <> cFirstWord ;
           AND "FUNCTION" <> cFirstWord) ;
           OR EMPTY(cFirstWord)
         nProcLine = nProcLine - 1
         cCodeLine = LTRIM(STRTRAN( ;
            aCodeByLines[nProcLine], CHR(9), " "))
         nHeaderEnds = AT(" ", cCodeLine)
         cFirstWord = UPPER(LEFT( cCodeLine, ;
            IIF(nHeaderEnds=0, LEN(cCodeLine), ;
            nHeaderEnds -1 )))
      ENDDO

      cProcName = ALLTRIM(SUBSTR( ;
         aCodeByLines[ nProcLine ], nHeaderEnds + 1))
      nParmsBegin = AT("(", cProcName )
      IF nParmsBegin <> 0
         cProcName = ALLTRIM(LEFT( cProcName, ;
            nParmsBegin - 1))
      ENDIF

      * Compute code line within procedure
      nCodeLine = nCodeLine - nProcLine + 1
      IF lDropNameLine
         nCodeLine = nCodeLine - 1
      ENDIF
   ELSE
      cProcName = ""
   ENDIF

   This.AddMatch(cFileName, cObjName, ;
                 cProcName, nCodeLine)
```

```
   * Now find next match
   nMatchPos = AT(This.csearchstring, cContents, ;
                  nMatchCount + 1)
ENDDO

RETURN
```

SearchField looks for the search string in the code. Then, the method searches backward from the line containing the search string to find the beginning of the containing method. The AddMatch method is called to store the information in the aMatches array. That method is quite simple, so I won't show it here.

In VFP 7, the new EditSource() function makes it easy to open a file at a particular line. To demonstrate the technique, the class has a very basic OpenMatches method that just runs through the aMatches array, opening each file. Because it doesn't check whether the file is already open, each method is positioned at the last match it contains.

This project search class is really meant as a proof of concept. It has no user interface and doesn't address many of the issues you'd want, such as case-sensitivity, full word matches vs. contained strings, and so forth.

Fortunately, someone else has already done the hard job of getting all those things to work. Steve Dingle has created a public domain tool for searching in a project. It handles all these issues, has an intuitive user interface, and is configurable. He's currently working on making the tool's search capabilities available programmatically. The current version of Project Search is included on this month's PRD.

So, while the short answer to your question is "you can't do that," the true answer is that once again, VFP's extensibility and open architecture make it possible to go far beyond what's built into other tools.

–Tamar