Make Thor Your Own

It's easy to add your own tools to Thor and to take advantage of Thor's built-in capabilities.

Tamar E. Granor, Ph.D.

In my last two articles, I wrote about some of the tools that come with the VFPX project, Thor. This month, I show you how to add existing tools to Thor, and how to take advantage of the code that's included with Thor.

Thor comes with an incredible collection of tools that make developing software with VFP easier. If that were all it offered, it would be worth installing.

However, Thor is also a container for any tools you want to add. I have a number of small tools I've written at one time or another, mostly with no user interface. For example, one of them goes through a project and fills a cursor with the names of all the form classes used in that project (that is, those on which at least one form is based). The whole thing is about 40 lines of code. The problem with these little tools is that when I want to use them, I have to find them and look at the code to remember how. Then I have to either make sure the code is in the path or specify the full path in order to run the tool.

One of the design criteria for Thor was to make it easy for people to add tools and to share them. That way, you can take all the little tools like the one I describe above and stick them into the Thor Tools menu to keep them handy. Among other benefits, Thor manages the code so you don't have to worry about paths.

To add a tool to Thor, open the Thor configuration form (Thor | Configure from the menu) and click the Tool Definitions tab. Click the Create Tool button to open the Create Tool dialog, shown in **Figure 1.** Once you give the tool a name using the textbox preceded by "Thor_Tool_," click the Create button to open a template file for the tool. (The string "Thor_Tool" becomes part of the tool's name.)

Thor tools require a specific format, which is provided by the template. The default template is shown in Listing 1, reformatted to fit the page. The bulk of the template provides a place to give Thor information about this tool; to create a tool, fill in one or more of the properties listed. Prompt is required and contains the prompt that will appear on the Thor Tools menu. Description appears only in the Thor configuration dialog and Tool Launcher. When you search in Tool Launcher, both the prompt and the description are searched.

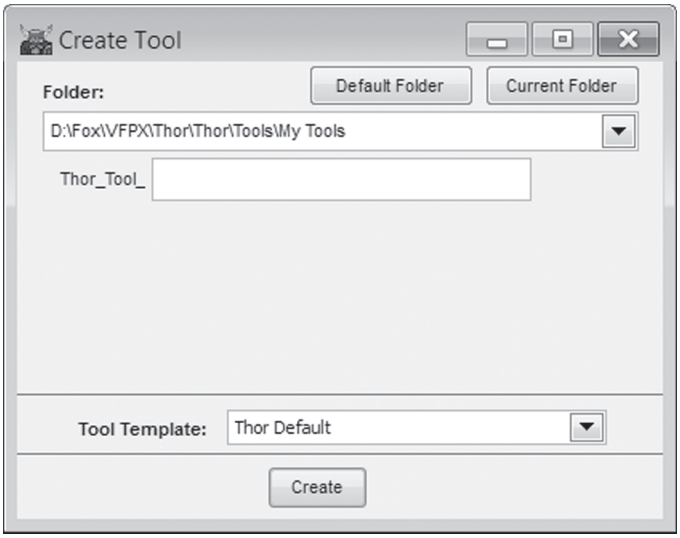

Figure 1. To add a new tool to Thor, specify the name of the tool in this dialog and click Create.

Listing 1. The default Thor template shows exactly what's required to create a Thor tool.

```
Lparameters lxParam1

*********************************************
*********************************************
* Standard prefix for all tools for Thor,
*   allowing this tool to
*   tell Thor about itself.

If Pcount() = 1              ;
    And 'O' = Vartype (lxParam1)  ;
    And 'thorinfo' == Lower (lxParam1.Class)

  With lxParam1

    * Required
    .Prompt       = 'Prompt for the tool'
                    && used in menus


    * Optional
    Text to .Description NoShow
Enter a description for the tool here
    EndText
    .StatusBarText = ''
    .CanRunAtStartUp = .T.

    * These are used to group and sort tools
    * when they are displayed in menus or the
    * Thor form
    .Source       = ''
      && where did this tool come from?  Your
      && own initials, for instance
    .Category      = ''
      && creates categorization of tools;
      && defaults to .Source if empty
```

```
    .Sort       = 0 && the sort order for all
      && items from the same Category

    * For public tools, such as PEM Editor,
    * etc.
    .Version     = ''
      && e.g., 'Version 7, May 18, 2011'
    .Author        = ''
    .Link          = ''
      && link to a page for this tool
    .VideoLink     = ''
      && link to a video for this tool

  Endwith

  Return lxParam1
Endif

If Pcount() = 0
  Do ToolCode
Else
  Do ToolCode With lxParam1
Endif

Return

*********************************************
*********************************************
* Normal processing for this tool begins here.
Procedure ToolCode
  Lparameters lxParam1

EndProc
```

The Category and Sort properties let you specify where the tool appears in the list of Thor tools. That list appears in the Thor Tools menu, in the Configuration form and in the Launcher. If Category is specified, the tool appears in that group; you can specify multiple levels in the menu by separating the items with the vertical bar ("|"). For example, to add an item to the Misc. group in the Code menu, specify "Code|Misc."

You might use your initials or your company name to group all of your own tools together.

The Source property, which can also be used for this purpose, is deprecated, so just leave that property empty.

The Sort property determines the position of this item in the specified submenu.

The last set of properties in the template is relevant only for tools being shared with the VFP community.

Listing 2 shows the properties set for my tool that gets a list of form classes used in a project. (The trailing comments for the properties have been removed to save space.)

Listing 2. The Thor properties set for the Get form classes tool.

```
    * Required
    .Prompt       = 'Get form classes'

    * Optional
    Text to .Description NoShow
Fill a cursor with names of the form classes
used in a project.
    EndText
    .StatusBarText = ''
    * These are used to group and sort tools
    * when they are displayed in menus
    * or the Thor form
    .Source       = ''
    .Category      = 'TEG'
    .Sort       = 0
```

The heart of the tool is the ToolCode procedure; that's where you put the code to perform the task. Listing 3 shows the code added to ToolCode for the Get Form Classes tool. As you can see, it's not terribly complex. It checks for an active project, and if one is found, a cursor is created to hold the list of form classes. The code then loops through the files in the project. When a form file is encountered, it's opened as a table, and the form-level record found. If we've seen this form class before, it just counts this instance. If this is a new form class for our list, a record is added to the FormClasses cursor. After the loop is complete, the cursor opens in a BROWSE window.

Listing 3. The ToolCode procedure for the Get Form Classes tool.

```
LOCAL cClassName, oFile, oProject, nOldSelect

IF TYPE("_VFP.ActiveProject") = "U"
  MESSAGEBOX("No active project", 0+48, ;
             "Get form classes")
  RETURN
ENDIF

oProject = _VFP.ActiveProject

nOldSelect = SELECT()

IF USED("FormClasses")
  USE IN SELECT("FormClasses")
ENDIF

CREATE CURSOR FormClasses ;
  (cClass C(30), nCount N(3))
INDEX on UPPER(cClass) TAG cClass

SELECT 0
FOR EACH oFile IN oProject.Files
  IF oFile.Type = "K"
    TRY
      USE (oFile.Name) ALIAS __Form
      LOCATE FOR UPPER(BaseClass) = "FORM"
      cClassName = __Form.Class
      IF SEEK(UPPER(m.cClassName), ;
              "FormClasses", "cClass")
        REPLACE nCount WITH nCount + 1 ;
          IN FormClasses
      ELSE
        INSERT INTO FormClasses ;
          VALUES (m.cClassName, 1)
      ENDIF
      USE IN DBF("__Form")
    CATCH
    ENDTRY

  ENDIF
ENDFOR

* Make sure last form was closed.
USE IN SELECT("__FORM")

SELECT FormClasses
BROWSE NOWAIT
```

Once you've specified the necessary properties and added code to the ToolCode procedure, save the program. It automatically gets saved in the right place with the right name.

To test your tool, either close the Thor configuration form, or click its Thor button. Either one refreshes menus and hotkeys. Once you do so, the new tool is included in the Thor Tools menu, as in Figure 2. In the other Thor forms that list tools (the Configuration form and the Launcher), your added tool shows up with a yellow background.

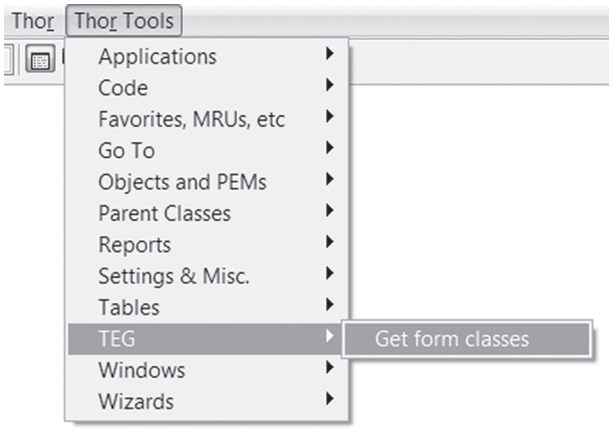

Figure 2. Once you finish the definition of a new tool and refresh Thor, the tool is shown on the menu.

The Thor Framework

While you can write a new tool with standard VFP code (as in the Get Form Classes tool), Thor offers a large library of capabilities that make it easier to write tools. The Thor Framework gives you access to several classes, as well as some standard items you may want.

To access the Thor Framework, choose Thor | More | Thor Framework from the menu. A window opens containing code you can cut and paste into your tool code. Figure 3 shows part of the Thor Framework. The Thor Framework is smart enough to show the correct path for your installation. (So Figure 3 shows where the files are located on my computer.) This means that if you have someone else's code that uses the Thor framework, you need to replace the LOCAL definition with the correct version for your installation.

A complete discussion of the Thor Framework is beyond the scope of this article, but I'll get you started with it and show a small example. (You'll find an overview at http://tinyurl.com/cney2dz. There's more information in the chapter "Creating Thor Tools" of "*VFPX: Open Source Treasure for the VFP Developer.*")

The Thor Framework is a set of classes, either built into Thor or built into other VFPX tools (like PEM Editor). The window that opens when you choose the Thor Framework gives you the code you need to access those classes. There are four core classes, shown in Table 1. There's a documentation page for each class on the VFPX site. The Thor Framework listing provides a link to the documentation.

Table 1. The Thor Framework provides access to a set of classes that simplify tool writing.

Class	Purpose
EditorWindow	Provides methods to access and modify IDE windows and to access and modify text in the current code editing window.
Tools	Provides methods (used by PEM Editor) to manipulate forms and classes.
ContextMenu	Provides methods used to create pop-up menus and sub-menus.
FormSettings	Provides methods to save and restore form properties, such as size and position.

As Figure 3 indicates, the way to use these classes is to instantiate them using an ExecScript() call to the Thor dispatcher, referenced as _Screen.cThorDispatcher. Save a reference to the instantiated object and call its methods.

thorframework.prg

```
************************************* External APPs ************************************

loInspector = ExecScript(_Screen.cThorDispatcher, "Class= inspector from inspector")

* editorwin home page = http://vfpx.codeplex.com/wikipage?title=thor%20editorwindow%20object
Local loEditorWin as Editorwin of "d:\fox\vfpx\thor\thor\tools\apps\pem editor\source\peme_editorwin.vcx"
loEditorWin = ExecScript(_Screen.cThorDispatcher, "Class= editorwin from pemeditor")

* tools home page = http://vfpx.codeplex.com/wikipage?title=thor%20tools%20object
Local loTools as Pemeditor_tools of "d:\fox\vfpx\thor\thor\tools\apps\pem editor\source\peme_tools.vcx"
loTools = ExecScript(_Screen.cThorDispatcher, "Class= tools from pemeditor")

************************************* Internal *************************************

lcVersion     = ExecScript(_Screen.cThorDispatcher, "Version=")

lcToolFolder  = ExecScript(_Screen.cThorDispatcher, "Tool Folder=")
```

Figure 3. The Thor Framework lets you take advantage of code in Thor in your tools.

The same technique, calling ExecScript() passing _Screen.cThorDispatcher as the first parameter, gives you access to some other features of the Thor Framework. Perhaps the most useful is the ability to find the path to any file that's in the Thor hierarchy; Listing 4 demonstrates.

Listing 4. You can use the Thor Framework to find the path to any file in the Thor hierarchy.

```
lcFileName    = ExecScript( ;
  _Screen.cThorDispatcher, ;
  "Full Path=" + m.cFileName)
```

The Get Form Classes tool described earlier in this article works only when you have a project open. One of the cool features of many Thor tools is that they can see what's at the cursor position and operate on that item. That would be a handy capability for this tool—if there's no open project, then find the name at the cursor position and attempt to open that project.

To figure out how that capability was provided, I poked around in the code for Thor tools that have it. To see how any Thor tool is implemented, open the Thor Configuration form and switch to the Tool Definitions page. Select the tool you're interested in the treeview and click the Edit Tool button, indicated in Figure 4. The form shown in Figure 5 appears. If all you want to do is see how the tool works, choose the second button, "View this file in Read-Only mode."

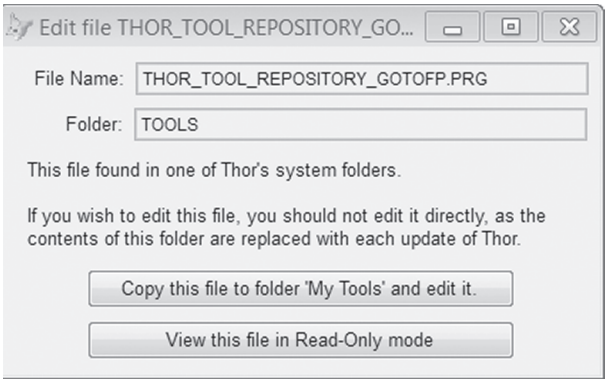

Figure 5. This form appears when you attempt to open any built-in Thor tool. To modify the tool, choose the first button. To simply look at its code, choose the second button.

In the code for the SuperBrowse tool, I found the lines in Listing 5. As you can see, it uses the Tools class from the Thor Framework.

Listing 5. The code that implements the SuperBrowse tool uses this code to determine what table to browse.

```
* tools home page =
* http://vfpx.codeplex.com/
wikipage?title=thor%20tools%20object
loTools = Execscript ( ;
  _Screen.cThorDispatcher, ;
  'class= tools from pemeditor')
loTools.UseHighlightedTable ( ;
  Set ('Datasession'))
```

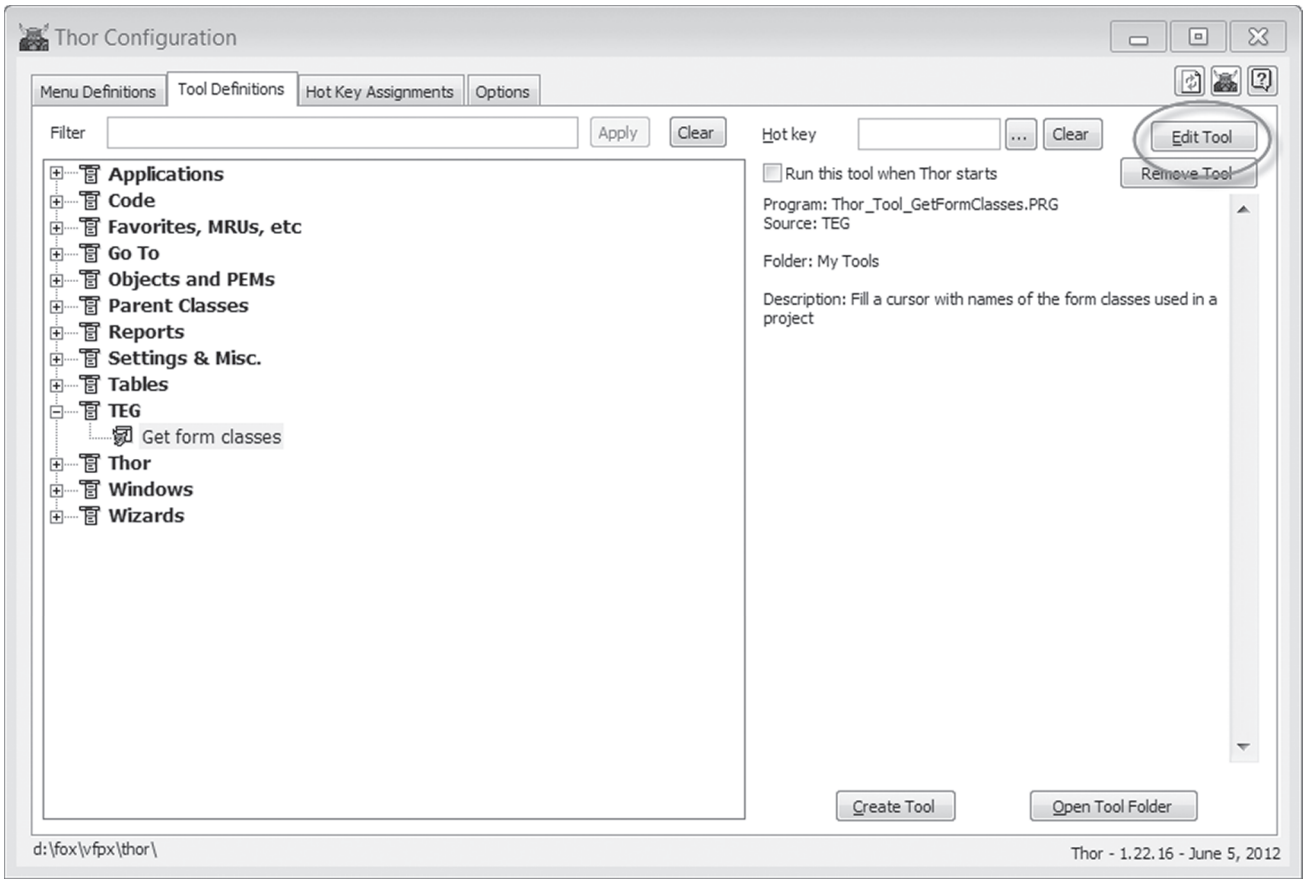

Figure 4. You can modify a tool by locating it in the Thor Configuration form and clicking Edit Tool.

I dug into the PEM Editor's source to find the UseHighlightedTable method, which I found in the PemEditor_Tools class of PEME_Tools.Vcx (which is the implementation of the framework's Tools class). Eventually, I found the line of code in Listing 6. Since the method name implies that it only grabs the highlighted text, I tested to confirm that the method, in fact, picks up the entire word where the cursor is positioned.

Listing 6. This line of code, used by the SuperBrowse tool, reads the text under the cursor.

```
lcAlias = ;
This.oUtils.oIDEx.GetCurrentHighlightedText()
```

The next step was to change the code for my Get Form Class tool. I opened the tool code as described above.

I copied the three lines from the framework that instantiate the Tools class and pasted them into the appropriate place in the ToolCode procedure. Once the Tools class is instantiated, I can use it to get the word under the cursor, and then try to open a project with that name. The relevant portion of the modified ToolCode procedure is shown in Listing 7. (Note that a couple of lines wrap here.)

Listing 7. Replace the code to check whether a project is open with this code to allow the Get Form Classes tool to work on the project whose name is under the cursor.

```
IF TYPE("_VFP.ActiveProject") = "U"
  * tools home page = http://vfpx.codeplex.
com/wikipage?title=thor%20tools%20object
  Local loTools as Pemeditor_tools of ;
    "d:\fox\vfpx\thor\thor\tools\apps\pem edi-
tor\source\peme_tools.vcx"
  loTools = ExecScript( ;
  _Screen.cThorDispatcher, ;
  "Class= tools from pemeditor")

  lcText = ;
    loTools.oUtils.oIDEx.;
    GetCurrentHighlightedText()

  lProjWasOpen = .F.

  TRY
    MODIFY PROJECT (lcText) NOWAIT
    lSuccess = ;
      (TYPE("_VFP.ActiveProject") <> "U")
  CATCH
    lSuccess = .F.
  ENDTRY
ELSE
  lSuccess = .T.
  lProjWasOpen = .T.
ENDIF

IF NOT m.lSuccess
  MESSAGEBOX("No active project", 0+48, ;
             "Get form classes")
  RETURN
ENDIF
```

In testing, I found that this code works only when the specified project is in the path. It turns out that there's a better way to do this (which wasn't available when I first looked at the problem).

Thor Procs

In addition to the Thor Framework, Thor also includes a set of programs called Thor Procs. These live in the Tools\Procs folder of your Thor installation and have names that begin "Thor_Proc_." The rest of the name describes the functionality.

To use a Thor Proc, call it using an EXECSCRIPT() call, as in Listing 8.

Listing 8. Thor Procs are called using the EXECSCRIPT() notation.

```
ExecScript(_Screen.cThorDispatcher, ;
           'Thor_Proc_SomeThorProc', ;
           'MyParameter')
```

The Thor Proc GetHighlightedText returns the text at the current cursor position, and can handle filenames including a path. This provides a better solution to allow the Get Form Classes tool to work by just clicking into the name of the project. The Proc accepts a single parameter to indicate what to return; Table 2 shows the acceptable values for the parameter.

Table 2. Pass a string to Thor_Proc_GetHighlightedText to indicate what text to return.

Parameter value	Returns
Empty (omitted)	The currently highlighted text.
"Name"	The word the cursor is currently positioned in.
"File Name"	The filename the cursor is currently positioned in, including path.
"Object Name"	The object name the cursor is currently positioned in, including any objects specified by WITH statements.

Listing 9 shows the revised part of the code that handles opening the file, if necessary.

Listing 9. Using the Thor Proc GetHighlightedText, we can pick up the full name and path of the project in the Get Form Classes tool.

```
IF TYPE("_VFP.ActiveProject") = "U"
  lcText = ;
    ExecScript(_Screen.cThorDispatcher, ;
             "Thor_Proc_GetHighlightedText", ;
             "File Name")

  lProjWasOpen = .F.

  TRY
    MODIFY PROJECT (lcText) NOWAIT
    lSuccess = ;
      (TYPE("_VFP.ActiveProject") <> "U")
  CATCH
    lSuccess = .F.
  ENDTRY
ELSE
  lSuccess = .T.
  lProjWasOpen = .T.
ENDIF
```

```
IF NOT m.lSuccess
  MESSAGEBOX("No active project", 0+48, ;
             "Get form classes")
  RETURN
ENDIF
```

Add Your Tools

While the Get Form Classes tools isn't something I can imagine using often enough to bother adding it to Thor, the ability to make all those little bits of code I've written over the years easily accessible is very powerful. It increases the odds that I'll reuse these tools. If they're easier to use, I'm more likely to invest time improving them, as well. No doubt you have some of these little tools floating around too; Thor offers you a way to make them a lot more useful.

In my next article, I'll look at Thor's structure for providing options for tools, so they can work differently in different situations or for different users.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of nearly a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro *and* Taming Visual FoxPro's SQL. *Her latest collaboration is* VFPX: Open Source Treasure for the VFP Developer. *Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.*