To Boldly Go Where No VFP'er has Gone Before: The Tools of Thor

Jim Nelson
The Kong Company
21 Los Vientos Drive
Newbury Park, CA 91320
Voice: (805) 498-9195
Email: JimRNelson@GMail.Com

This document explores the recent explosion of tools for the IDE: tools that you always thought should have been part of FoxPro, tools that you have always wanted – and more. These tools come from two distinct sources: PEM Editor 7 w/IDE Tools, which has long outgrown PEM Editor's original intent of simply managing properties and methods, and the Thor Repository, a collection of tools created by members of the FoxPro community.

Thor is the first of these tools to be described, as it manages how all of the other tools are accessed. It provides a user interface to control the assignment of hot keys and developer-defined menus for tools, a mechanism for creating new tools, a framework on which new tools can be built, and a structure that simplifies sharing these tools between developers.

Section 1. Introduction

There are a number of valuable tools that are available in other development environments, but are sadly lacking in FoxPro's own IDE. For some time, there has been a considerable effort afoot to expand the list of IDE tools by reproducing (where possible) the tools from other IDEs (such as Visual Studio), by building on and enhancing existing tools, and by creating brand new tools appropriate to FoxPro.

Thor is one of these tools, but the most visible, as it manages how all of the other tools are accessed. It provides a user interface to control the assignment of hot keys and developer-defined menus for tools, a mechanism for creating new tools, a framework on which new tools can be built, and a structure that simplifies sharing these tools between developers.

The UI provides four different methods for accessing these tools:

- By assigning hot keys to them
- By creating pop-up menus accessible by hot keys
- By adding them as bars under any of the VFP system pads (File, Edit, View, etc)
- By creating new pads in the VFP system menu and adding the tools as bars under these new pads.

Unlike the normal limited set of hot keys available from ON KEY LABEL, Thor provides for the full range of multiple-keystroke combinations (Ctrl + Alt + A, for instance).

This document begins by showing how to install and start using Thor, how to use its UI to assign hot keys and create menus and submenus, and how to use Thor to simplify access to your own private tools.

Using Thor as the backbone for the remainder of the document, we will explore the published tools accessible in Thor (as mentioned in the first paragraph). The result of the work mentioned here is a suite of tools that provides a large number of features never before available in VFP. You will find that these tools will rapidly become indispensable in your day-to-day programming life.

Section 2. Thor

Section 2.1. Overview

Thor is a tool for managing add-on tools in the IDE.

It provides four different methods for accessing these tools:

- By assigning hot keys to them
- By creating pop-up menus accessible via hot keys
- By adding the tools as bars under any of the VFP system pads (File, Edit, View, etc.)
- By creating new pads in the VFP system menu and adding the tools as bars under these new pads.

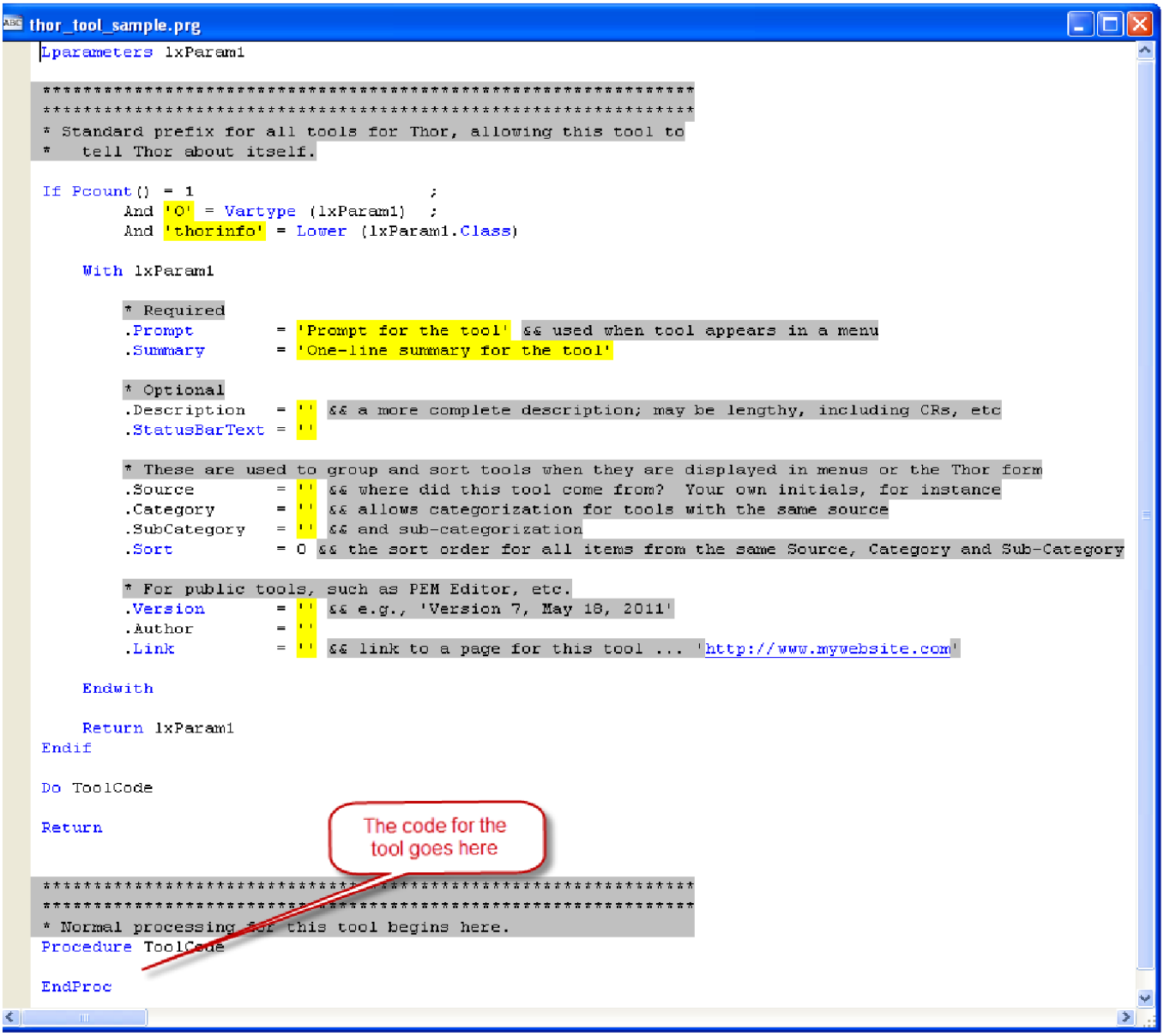
There are four parts to Thor:

- The catalog of tools managed by Thor. These are simply PRGs with a particular structure.
- The underlying free tables, which contain the definitions of the menus, hot key assignments, etc.
- The Thor form, accessible by either hot key or from a menu pad on the VFP system menu, which manages all the menu and hot key definitions stored in the tables.
- An APP which takes the definitions from the tables and updates the VFP system menu (if appropriate), creates the pop-up menus, and assigns the hot keys to the pop-up menus and/or individual tools. Since these definitions survive a CLEAR ALL, this need be run only once at the beginning of an IDE session.

Section 2.1.1. What are "Tools"?

Tools are PRGs that follow a particular structure so that Thor can recognize them.

The "header" of each tool is a group of about 40 lines which act as a questionnaire, allowing the tool to tell Thor about itself. When creating a new tool when using the Thor form, it starts out looking like this:



The actual code for the tool follows this header. As always, it is advantageous to browse other tools (something you can also do within the Thor form), to see examples of how these properties are normally used.

The process for creating a new tool (when using the Thor form) will guide the creation of the tool into a folder that Thor recognizes, either the default folder new tools (**Thor\Tools\My** Tools), or any folder in the path.

Suggestion: assign the same value for *.Source* for all of your personal tools, so that the Thor form will group them together.

Section 2.1.2. Where You Can Get Tools

There are three sources of ready-made tools for Thor, all of which can be downloaded from VFPx: 'PEM Editor w/ IDE Tools', the 'Thor Repository', and 'GoFish'. Thor was actually created so that it could be possible to publicize and make available the tools from these sources. Each will be explained in detail in the rest of this document.

Section 2.1.3. About Hot Keys

Thor provides for a new capability not previously available -- the ability to assign hot keys which use two or three of the Shift, Ctrl, and Alt keys, such as {Ctrl + Alt + A}. (Actually, using all three requires quite some manual dexterity.)

Keyboard macros (using FKY files) are the only VFP feature which recognize usage of two or three of these keys, but macros do not cause program execution, they only place characters into the keyboard buffer.

Section 2.2. Installing Thor

Thor must be installed in a permanent folder; it creates some folders and tables which must always be available. It is suggested that it be installed in a folder that you regularly back up.

Download the current version of Thor from:

http://vfpx.codeplex.com/releases/view/67395

There are two different strategies for selecting an installation folder for Thor:

- Install it in a common folder (such as in your path), so that Thor.App can be easily accessed.
- Install it in its own separate folder, and then use RunThor.PRG (see below) to access Thor.App

After you have downloaded the Zip file into its installation folder, do the following:

```
Clear All
Do Thor.APP
```

This will

- create a folder named Thor in the installation folder
- create some sub-folders and files in that folder

- update the VFP system menu (by adding a menu pad for Thor)
- and open the Thor form.

Note the installation does not affect VFP in any other way (it does not SET any variables, modify foxcode, etc.) and may safely be repeated as many times as desired.

Note also that this need not be repeated with any new versions of Thor, as the installation process is performed automatically when a newer version of Thor is used.

Section 2.3. Running Thor

Thor needs to be started once each IDE session. (It survives Clear All). It can be done in either of two ways:

(a) By calling Thor.App directly

```
Do Thor.APP with 'Run'
```

(b) By calling RunThor.PRG, a PRG which is created in the Thor sub-folder and which can be copied anywhere (such as to a folder in the path)

```
Do RunThor
```

Note: The distinction here is that Thor.APP cannot be moved from its installation folder (if so, it has to be re-installed). RunThor.PRG can be moved, as it contains an explicit reference to the folder where Thor.APP is installed.

Section 2.4. Installing tools published on VFPx

There are a number of tools that can be downloaded directly from VFPx:

Section 2.4.1. Tools from PEM Editor

Version 7 of PEM Editor, now re-named 'PEM Editor w/ IDE Tools', contains more than three dozen tools that can be accessed through Thor. These include some tools released in version 6 of PEM Editor, along with a large number of completely new tools. These can be downloaded from the PEM Editor page.

PEM Editor also "publishes" a pair of objects that simplify building further tools. More than half of the original tools in the Thor Repository use these objects.

Registration of the tools from PEM Editor occurs automatically (if Thor is running) when you install PEM Editor by executing:

```
Do PEMEditor.APP
```

Download the current version of PEM Editor from here:

http://vfpx.codeplex.com/releases/view/67528

Section 2.4.2. The Thor Repository

Inherent in the design of Thor is the anticipation that members of the FoxPro community will have utilities of value to be shared throughout the community. The structure of the tool PRGs make such sharing simple.

The 'Thor Repository' is a catalog of such tools. The intent is that this repository grow over time, as developers submit tools to be included. At the time this document was written, the repository had about twenty such tools.

All FoxPro'ers are encouraged to submit IDE tools that they think would be of value to the FoxPro community at large (or, for that matter, to particular sub-sets of the FoxPro community).

Tools can be submitted to this address: VFPThorRepository@GMail.Com

Tools will be accepted into the Repository if they meet some minimum standards, which include:

- They must act as designed and described.
- They must have no undesirable side effects.
- They must not make assumptions about file locations (such as folder names, other than system folders)
- They must follow the standard Thor-Tool format for a PRG, and their description must be as complete as possible.

To register the tools from the Repository with Thor, unzip the ZIP file into the folder Thor\Tools.

Download the current version of the Thor Repository from here:

http://vfpx.codeplex.com/wikipage?title=Thor%20Repository

Section 2.4.3. GoFish

GoFish 4 is an advanced code search tool for fast searching of Visual FoxPro source code. It can be integrated with Thor if Thor has been run in the current IDE session.

Download the current version of GoFish from here:

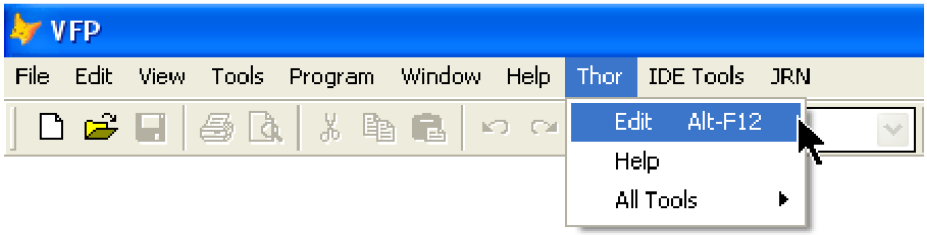
http://vfpx.codeplex.com/releases/view/71607

Section 2.5. The Thor Form

Section 2.5.1. Opening the Form

The Thor form can be opened a number of different ways:

- From the VFP system menu:



-
- By using the dedicated hot key. By default, this is Alt+F12, but it can be changed on the Options page
- By executing

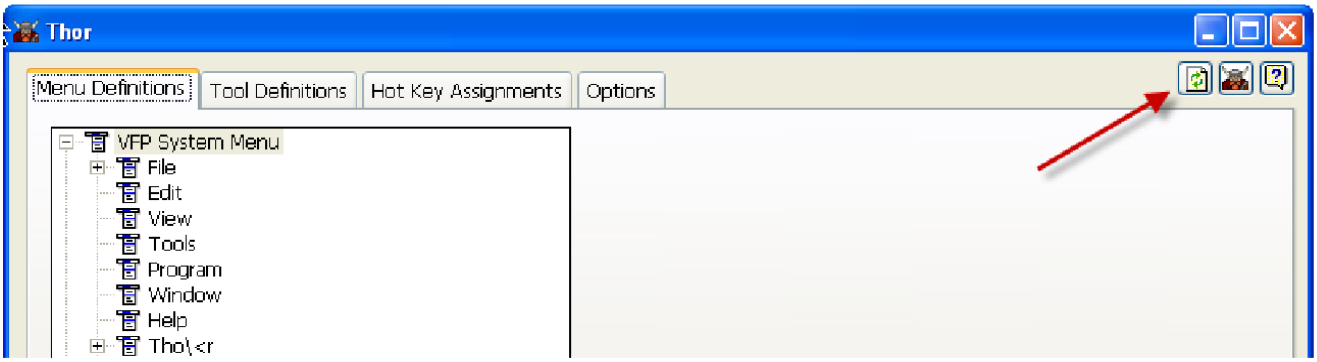
```
Do Thor.App with 'Edit'
```

Section 2.5.2. General Usage Notes

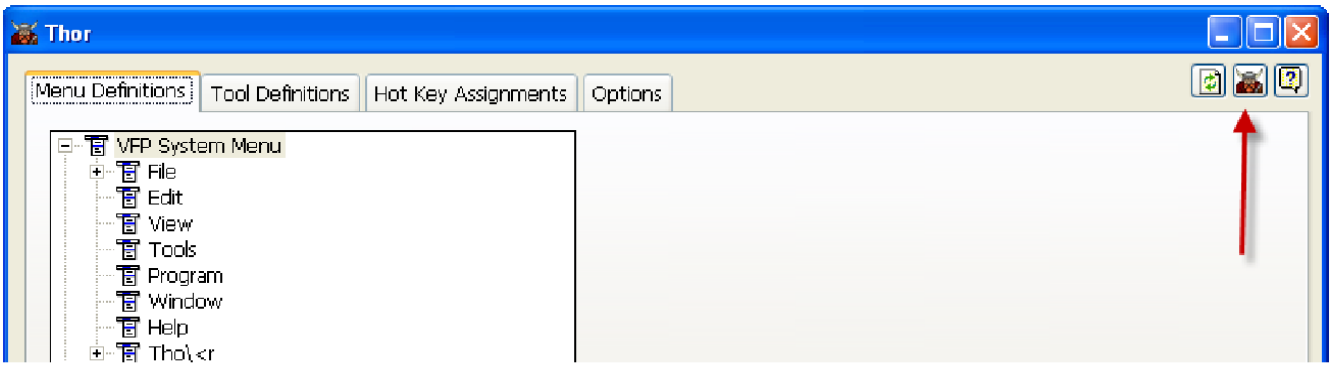
There are a few very important considerations to keep in mind when using the Thor form:

There are no 'Save', 'Cancel', or 'Undo' buttons on the form. Any changes made on the form are posted to the underlying tables immediately, and there is no provision for canceling or un-doing any changes.

When the form is opened, it reads the catalog of all tools and uses that catalog from there on. Thus, changes made to tools, or tools that are added while the form is opened, are not automatically included in the catalog. To synchronize the form with the catalog of all tools (such as after creating a new tool or modifying an existing one), use the Refresh button:



Any updates made to the VFP system menu, pop-up menus, or hot key assignments are not applied until the form is closed. To synchronize the VFP system menu, etc, with the form, use the Thor button:

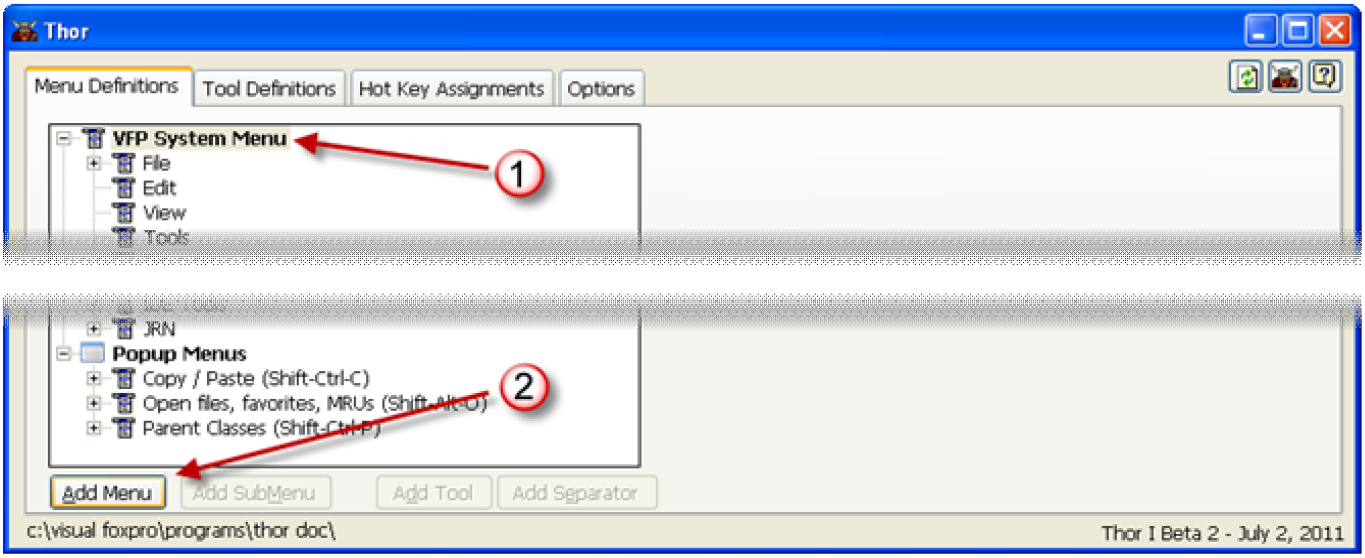


Section 2.5.3. Managing the VFP system menu bar

2.5.3.1. Creating new menu pads

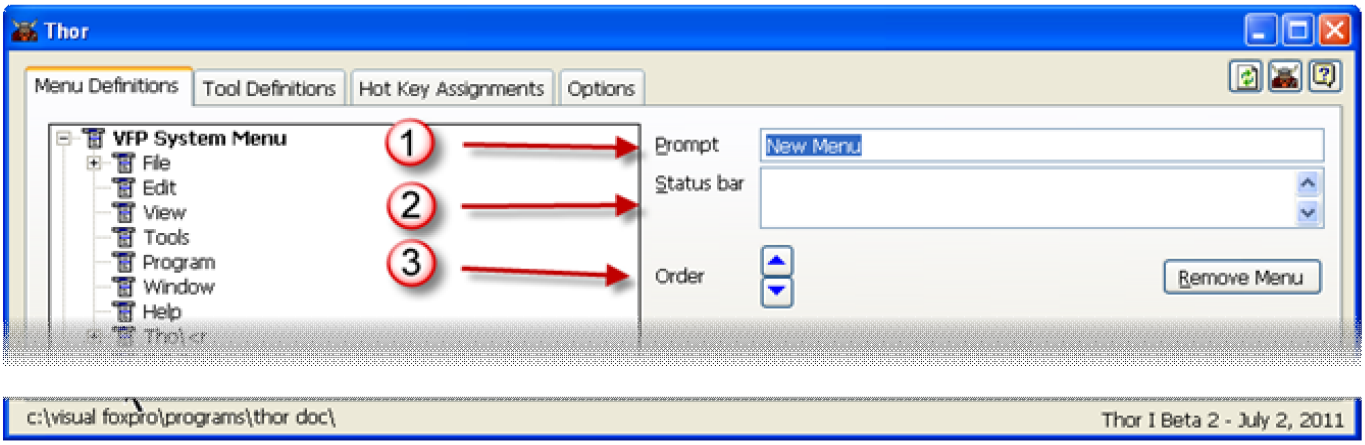
To create a new pad in the VFP system menu bar:

- click on the VFP System Menu node at the top of the TreeView
- then clicking on the Add Menu command button.



You can now:

- Assign the prompt for the new menu
- Assign the text that will appear in the status bar
- Move the new menu up to where you want it to be displayed.

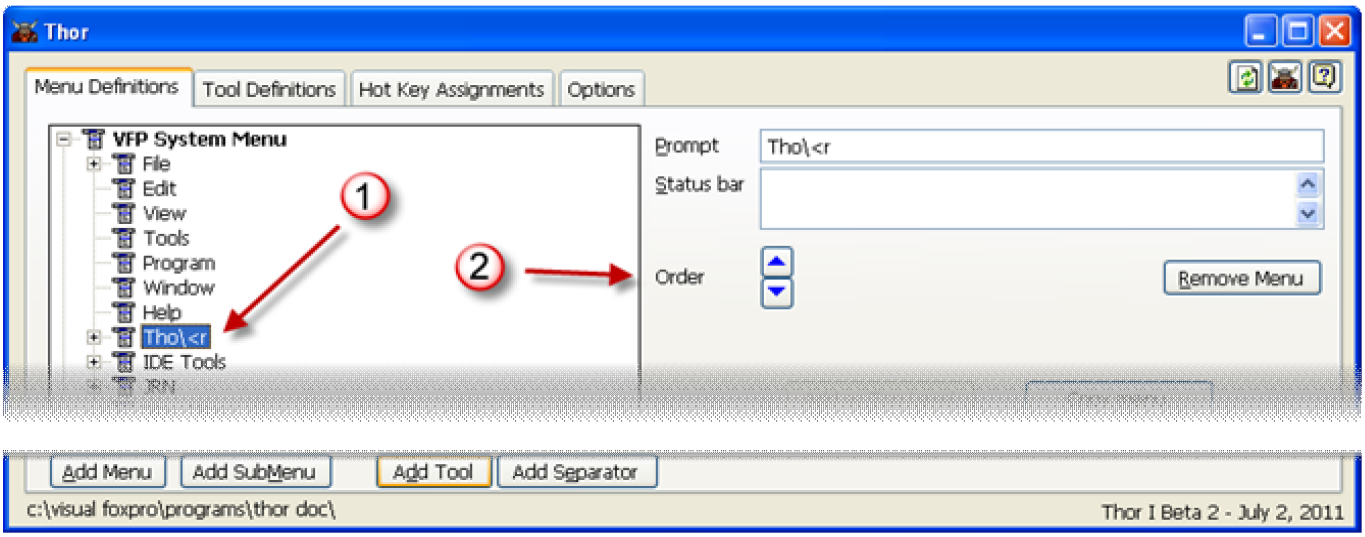


You can also take an existing menu (either a pop-up menu or a sub-menu) and add it to the VFP system menu bar.

2.5.3.2. Moving Menu Pads

You can move menu pads that you have created in the VFP system menu bar by:

- clicking on the menu pad you want to move
- then clicking on the up and down arrows



Notes:

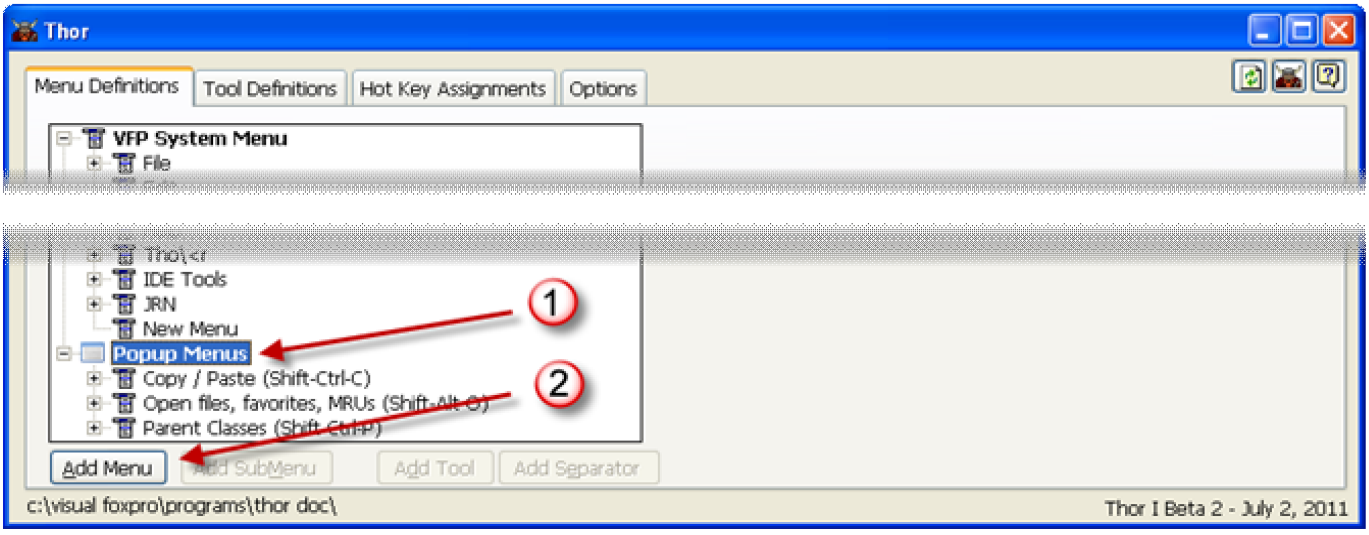
- You can not move the VFP-defined menu pads.
- The changes you made do not take effect immediately.

Section 2.5.4. Managing pop-up menus

2.5.4.1. Creating pop-up menus

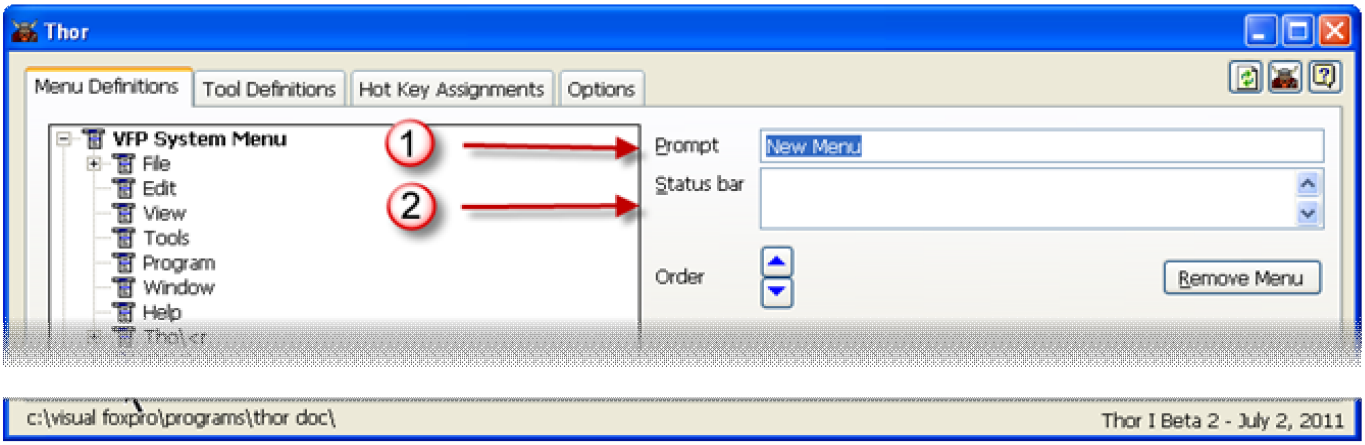
To create a new popup menu

- Click on the Popup Menus node in the TreeView
- Click on the Add Menu command button



You can now:

- Assign the prompt for the new menu
- Assign the text that will appear in the status bar



You can also take an existing menu (either a VFP system menu bar or a sub-menu) and make it available as a popup menu.

2.5.4.2. Assigning hot keys for popup menus

When you click on one of the popup menus, you can assign a hot key to it by using the controls on the right.

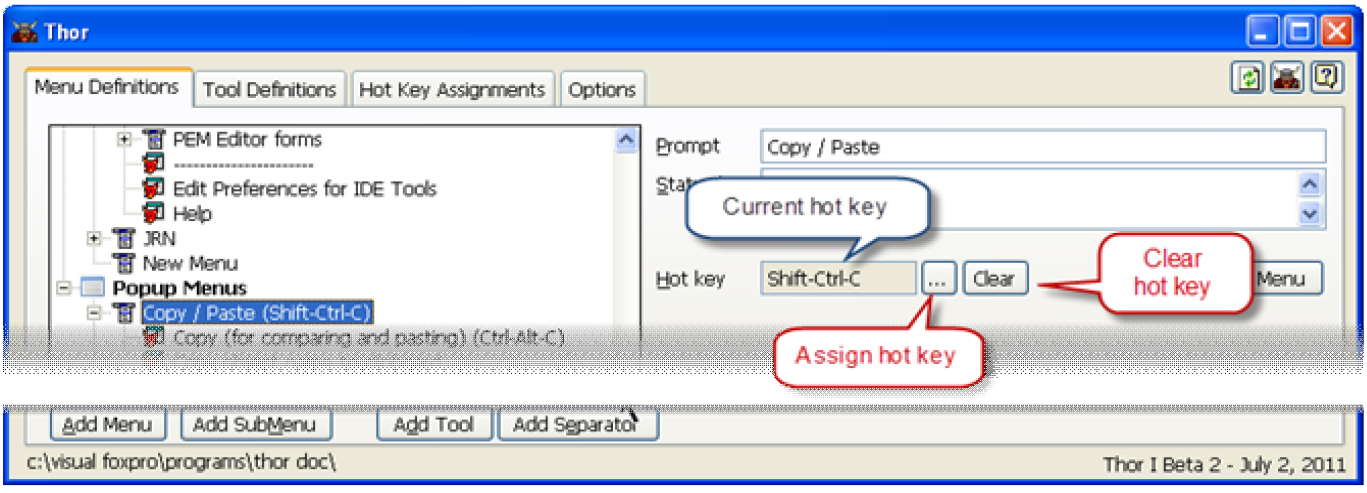
The hot key currently assigned to it appears in the disabled textbox. (If there is one already assigned, you can remove it by using the 'Clear' button.)

You can then click on the '...' button to be prompted for the new hot key. A small form will appear, requesting that you press the key combination to be used as the hot key. You can use any combination of Shift, Ctrl, and Alt.

Note that not all key combinations can be captured, and some of them will not be accepted because they are pre-empted by Windows or FoxPro itself (Alt-F10, for instance.)

You also are protected from assigning the same hot key to more than one tool.

To review all hot key assignments, including keyboard macros and On Key Label definitions, see the third page of the form. You can also change hot key assignments on that page.

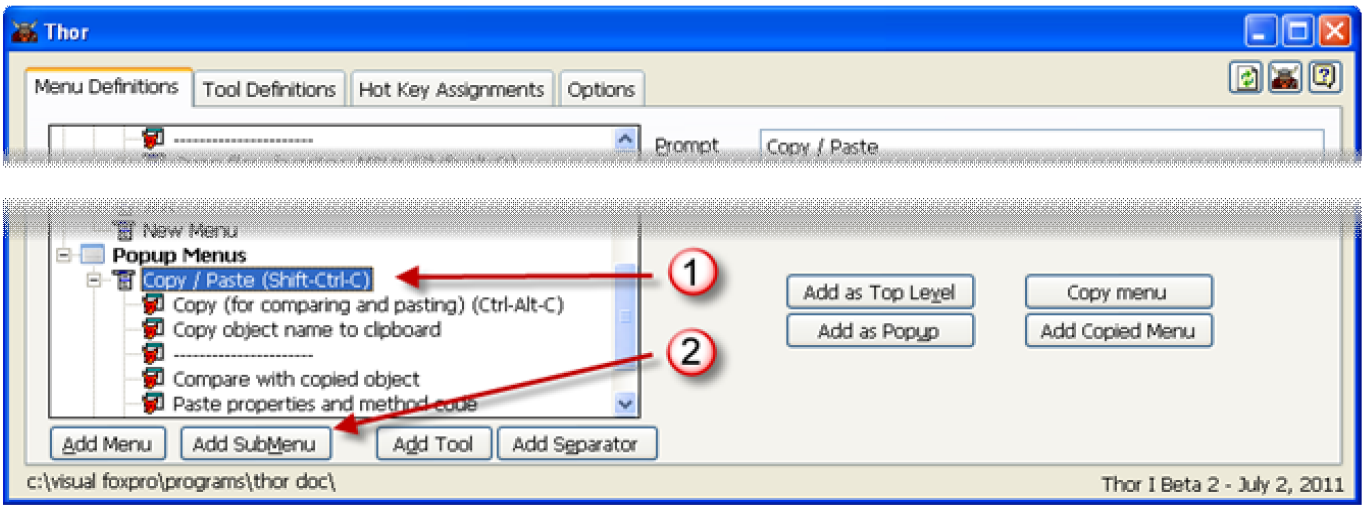


Section 2.5.5. Managing system menus, pop-up menus, and sub-menus

2.5.5.1. Creating sub-menus

To add a new sub-menu to any VFP system menu pad (including VFP's default menu pads), popup menu, or sub-menu:

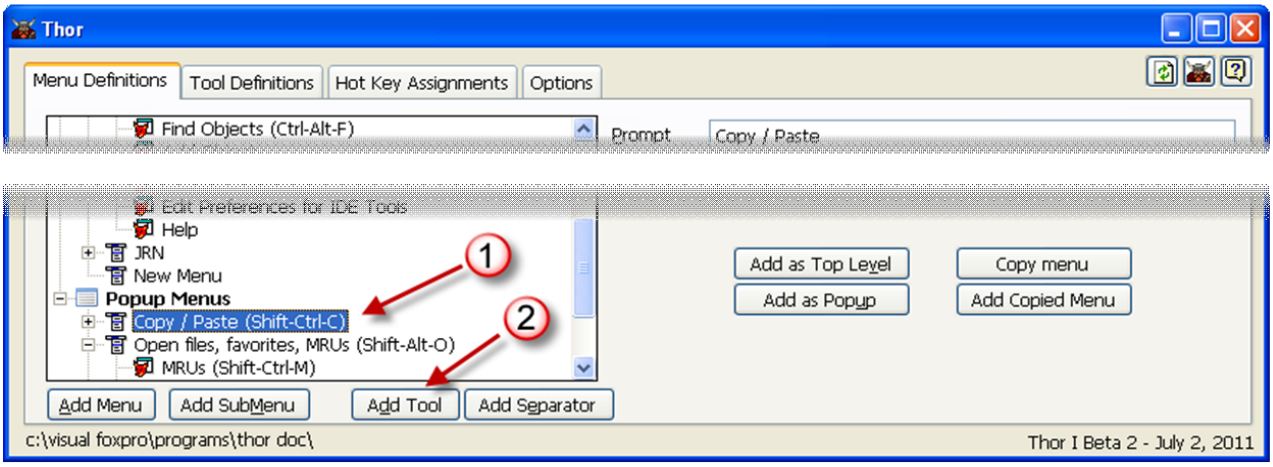
- Click on the menu to which you want to add the sub-menu
- Click on 'Add Sub-Menu'



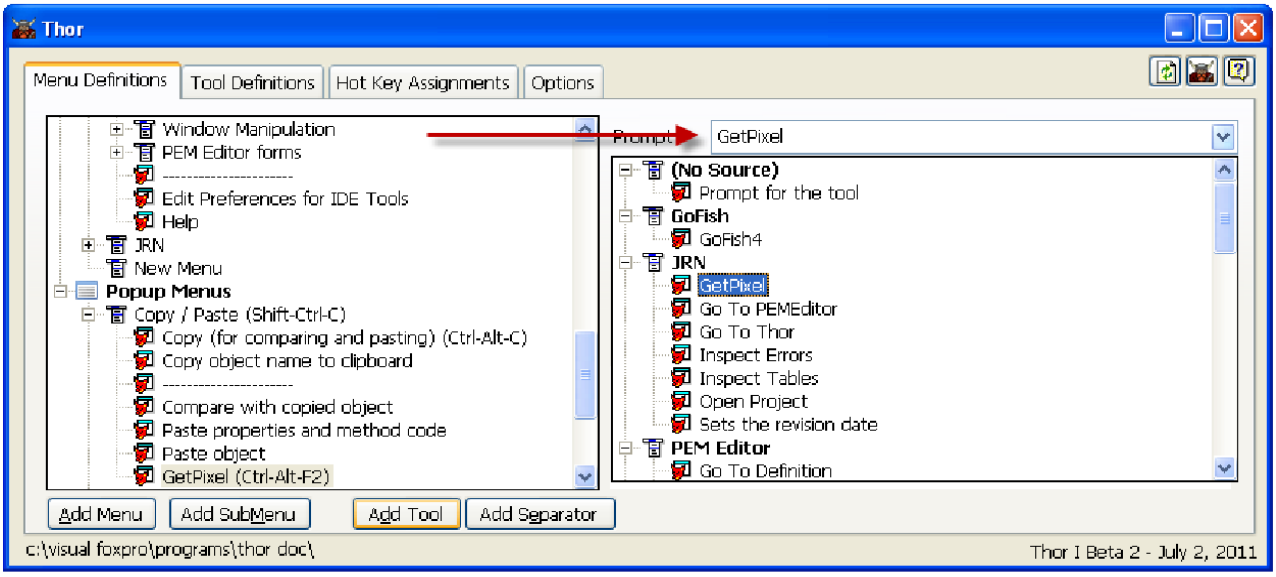
2.5.5.2. Adding tools to menus

To add a tool to any of the various kinds of menus:

- Click on the menu to which you want to add the tool.
- Click on 'Add Tool'



Then, drop-down the combobox to see the TreeView of all tools (organized by Source, Category, and sub-Category)

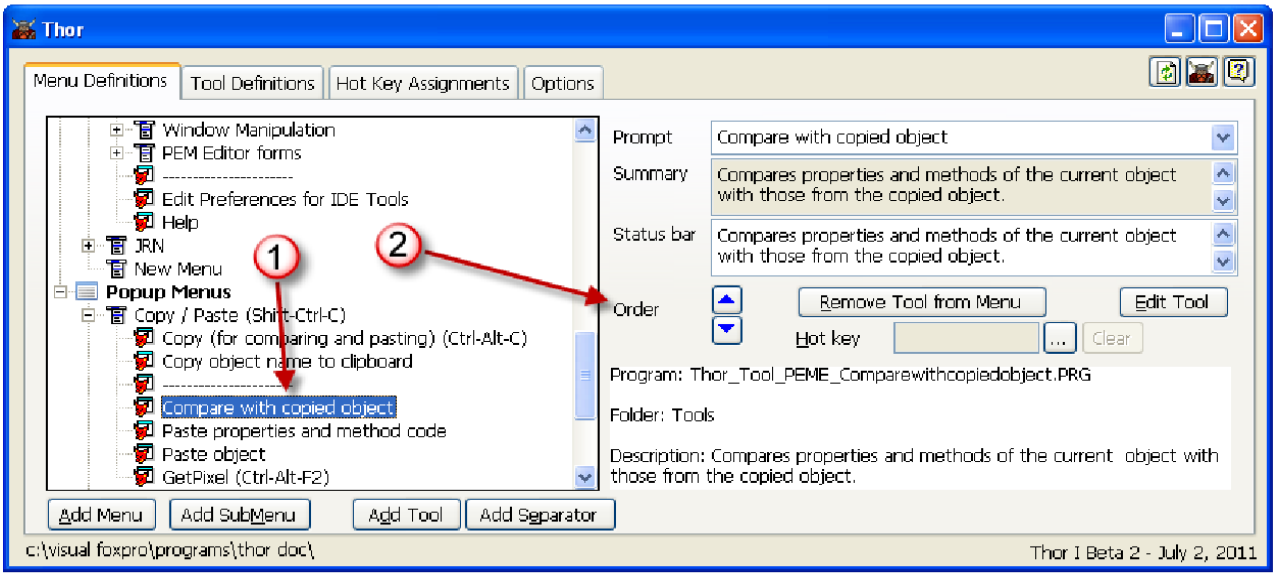


You can then change the prompt for the tool, if desired, and the status bar text.

2.5.5.3. Re-ordering tools and sub-menus within a menu

You can re-order the tools and sub-menus within a menu as follows:

- click on the tool or sub-menu
- use the up and down arrows



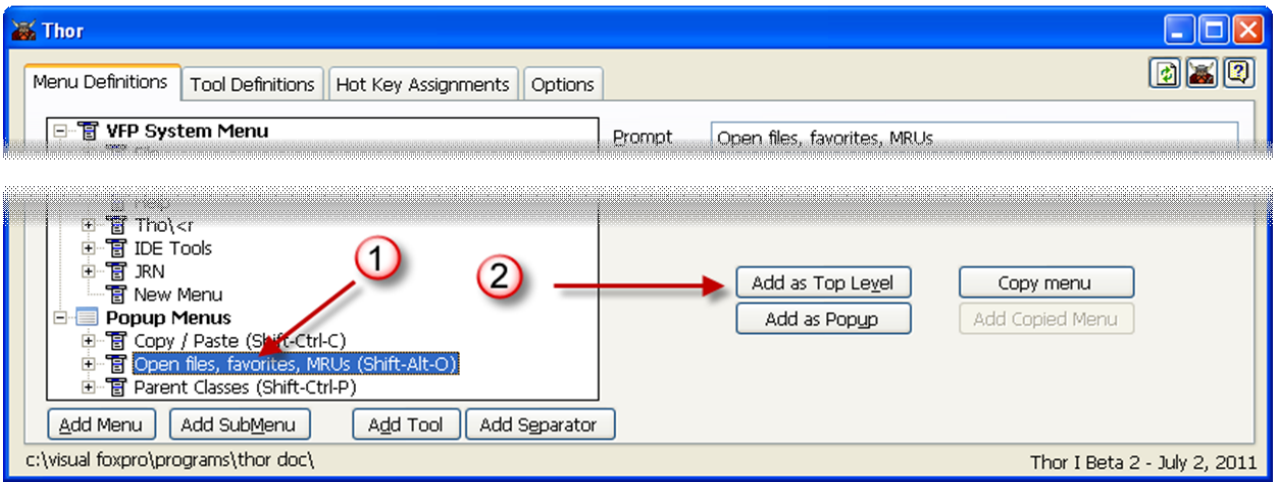
2.5.5.4. Duplicating menus

There are three different types of menus (VFP system menu pads, pop-up menus), and any menu you create can be used multiple times, as one or more of these three types.

Just to be clear, this means that the same menu can appear in multiple places. Internally, it will be the same menu, and any change made to it in one place will be replicated in all the other places as well.

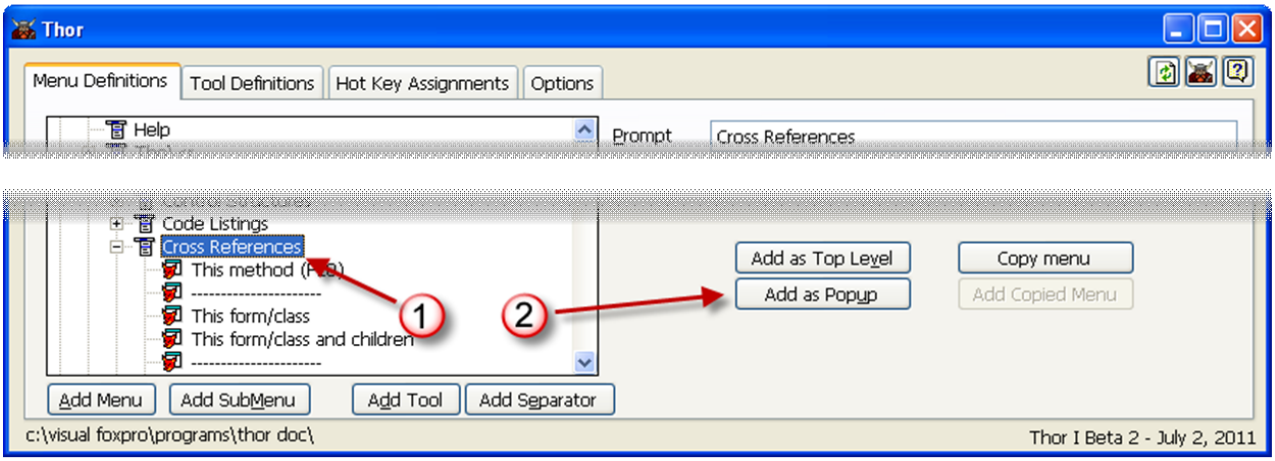
To make a popup menu or sub-menu into a menu pad on the VFP system menu bar

- Click on the menu (in the example below, a popup menu is chosen; it works the same for any sub-menu.)
- Click on 'Add as Top Level'



To make a VFP system menu or sub-menu into a Popup Menu

- Click on the menu
- Click on 'Add as Popup'



To make any menu into a sub-menu:

- Click on the menu you want to make into a sub-menu
- Click on 'Copy Menu'
- Click on the menu that you want to add the sub-menu
- Click on 'Add Copied Menu'

Section 2.5.6. Managing Tools

2.5.6.1. Browsing the list of tools

The second page (Tool Definitions) of the Thor form displays a grid showing all of the tools registered with Thor as well as what hot keys (if any) are assigned to them. It can also be used to assign hot keys, to edit existing tools, and to create new tools.

The grid on the left shows all the tools registered with Thor. grouped by their Source. Thus it is recommended that when you create your own tools, you assign the same Source to all of them, so they will all be grouped together there (and in other places in Thor.)

The second column shows the hot key that you have assigned to each tool, if any.

The third column shows the tools that you have chosen to be run each time that you run Thor. While this is in fact meaningless for most tools, it may be found desirable for those

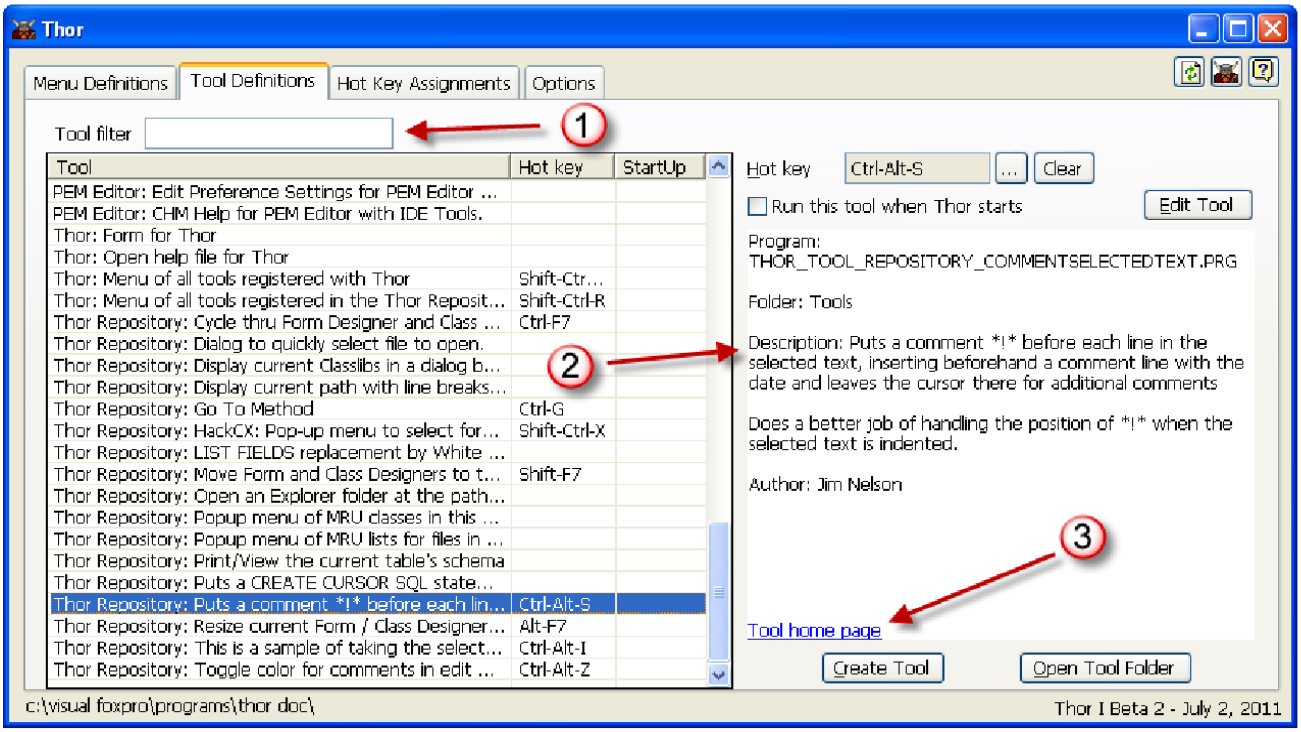
tools that bring up forms that you want to remain visible throughout your IDE session (such as PEM Editor, Document TreeView, or Object Size and Position)

There are a few other features to note when browsing this page:

You can use the Tool filter to narrow the list of tools shown. Note that the filter is only applied when you exit the field.

Most of the tools from PEM Editor and the Thor Repository have (or will have) more complete descriptions that appear in the large editbox on the right. A reasonable approach when starting as a new user of Thor, or when getting a new collection of tools from the Thor Repository, is to step through all the tools, reading the descriptions for each.

A number of tools, including most of the tools from PEM Editor, have (or will have) a link to the tool's home page for a thorough description of its use.



2.5.6.2. Assigning hot keys to tools

You can assign hot keys to tools on the second page (Tool Definitions) of the Thor form.

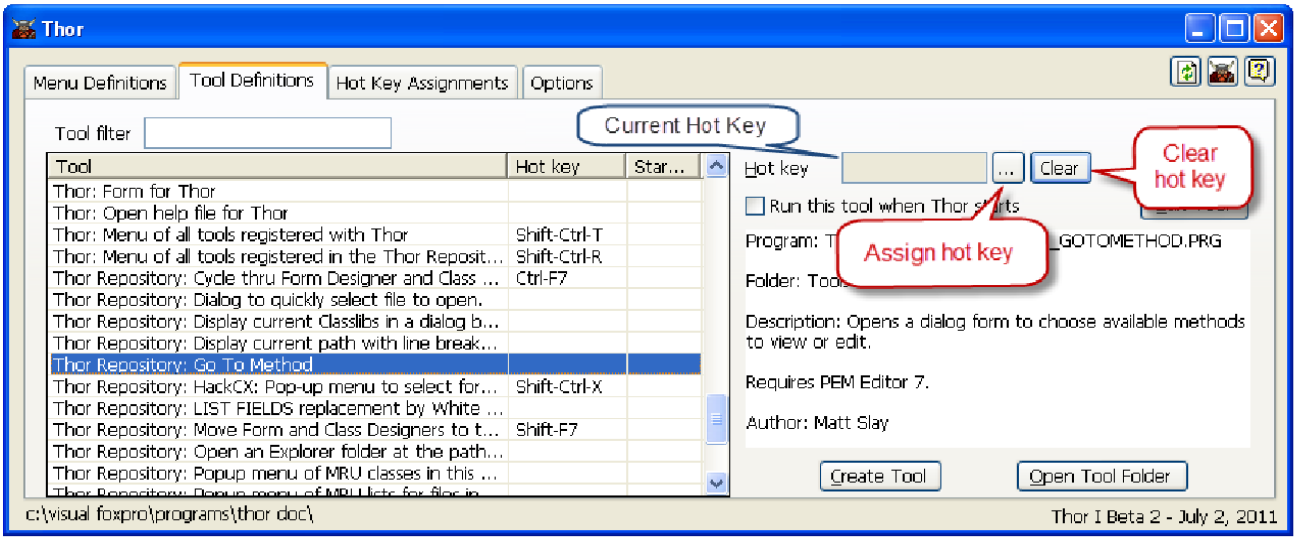
When you click on a tool, the hot key currently assigned to it appears in the control at the top right of the form. (If there is one already assigned, you can remove it by using the 'Clear' button.)

You can then click on the '...' button to be prompted for the new hot key. A small form will appear, requesting that you press the key combination to be used as the hot key. You can use any combination of Shift, Ctrl, and Alt.

Note that not all key combinations can be captured, and some of them will not be accepted because they are pre-empted by Windows or FoxPro itself (Alt-F10, for instance.)

You also are protected from assigning the same hot key to more than one tool.

To review all hot key assignments, including keyboard macros and On Key Label definitions, see the third page of the form (Hot Key Assignments). You can also change hot key assignments on that page.



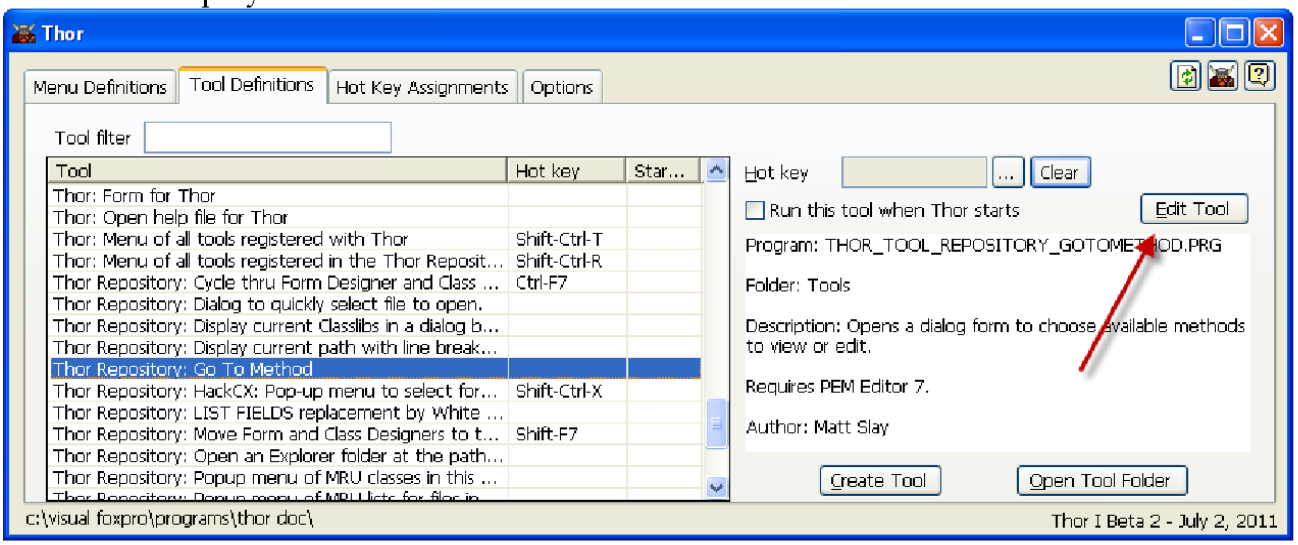
2.5.6.3. Editing Existing Tools

You can edit existing tools by using the second page (Tool Definitions) of the Thor form.

You can open a tool for editing either by double-clicking its row in the grid, or by selecting the row and then clicking the 'Edit Tool' button.

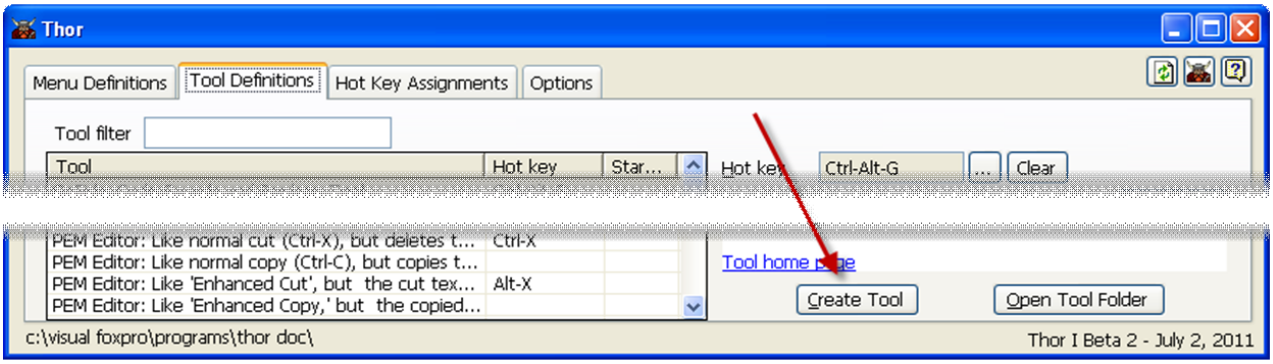
Note that changes made to the 'header' of the tool's PRG are not automatically reflected in

the form's display.



2.5.6.4. Creating New Tools

To create a new tool, use the 'Create Tool' button on the second page (Tool Definitions) of the Thor form.



This will bring up the 'Create Tool' form, which guides you in selecting the folder and file name for the new tool.

Thor look for files named Thor_Tool_*.PRG in the following folders:

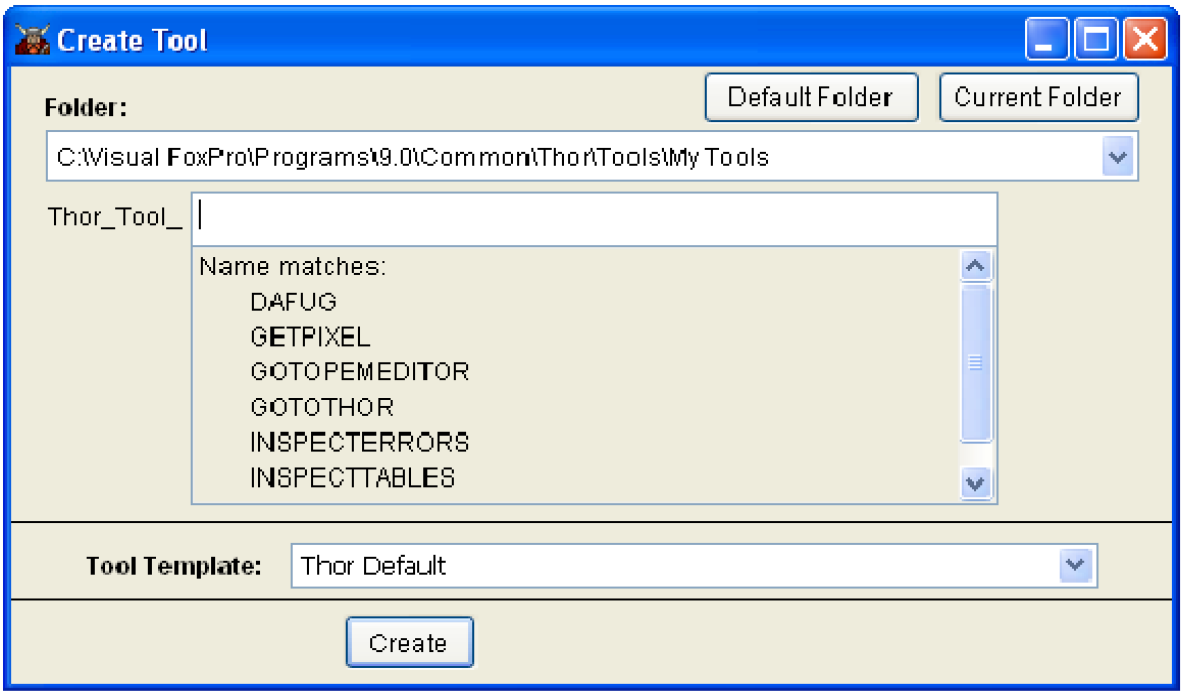
- CurDir()
- All folders in your path
- Thor\Tools\MyTools -- the recommended folder for personal tools
- Thor\Tools -- the recommended folder for downloaded tools, such as from PEM Editor 7 with IDE Tools or the Thor Repository

Obviously, there can be name conflicts, as there could be PRGs with the same name in different folders. Thor resolves these conflicts by recognizing the first such tool encountered, ignoring any following tools with the same name.

This design provides a substantial benefit, as it turns out. If the folder (Thor\Tools) is only used for downloaded tools, there is never any danger that your personal tools will conflict with them. Beyond that, you can select any of the downloaded tools, particularly any of those from the Thor Repository, make adjustments to them to suit your taste, and save them in Thor\Tools\MyTools. These personal copies will always take precedence, then, over any downloaded copies.

The 'Create Tools' form gives you a combobox showing all the folders that Thor will search (in the order they will be searched), as well as a textbox where you can enter the name of the tool you will be creating. The listbox underneath shows the names of all tools already in that folder whose names might be in conflict with the name you are creating.

The "Tool Template" combobox at the bottom of the form normally only contains one entry, for the Thor default template. You can create alternative templates for your own use by modifying this default template as you see fit, and then saving it as a PRG (any file name will do) in folder 'Thor\Tools\My Tools\Templates'



This will create the PRG and open it for you to edit.

The "header" of the new PRG is a group of about 40 lines which act as a questionnaire, allowing the tool to tell Thor about itself.

The actual code for the tool is to be placed in the Procedure 'ToolCode', at the very end of the listing.

As always, it is advantageous to browse other tools (something you can also do from within the Thor form), to see examples of how these properties are normally used.

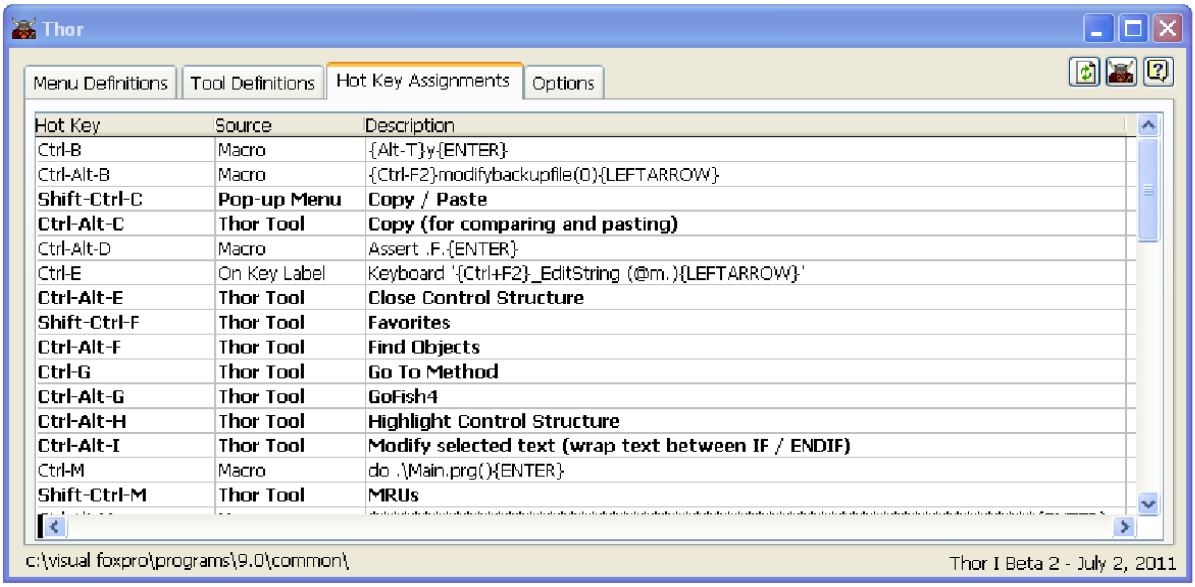
A final suggestion: assign the same value for .Source for all of your personal tools, so that the Thor form will group them together.

Section 2.5.7. Table of all hot keys, macros, and On Key Label assignments

The third page of the Thor form (Tool Definitions) displays a grid of all hot keys that have been assigned in Thor for popup menus and tools as well as all keyboard macros and On Key Label definitions. (While Thor does not manage the keyboard macros or On Key Label definitions, it is very handy to see all of these different definitions in one place.)

Clicking on a column header will sort by that column. Note that there are two different sorts available for the first column; consecutive clicks on the column heading will alternate between these two sorts.

For rows that are controlled by Thor (the bold rows), double-clicking on a row will allow you to re-define the hot key assigned. To remove the hot key assignment, use the right-click context menu on the row.



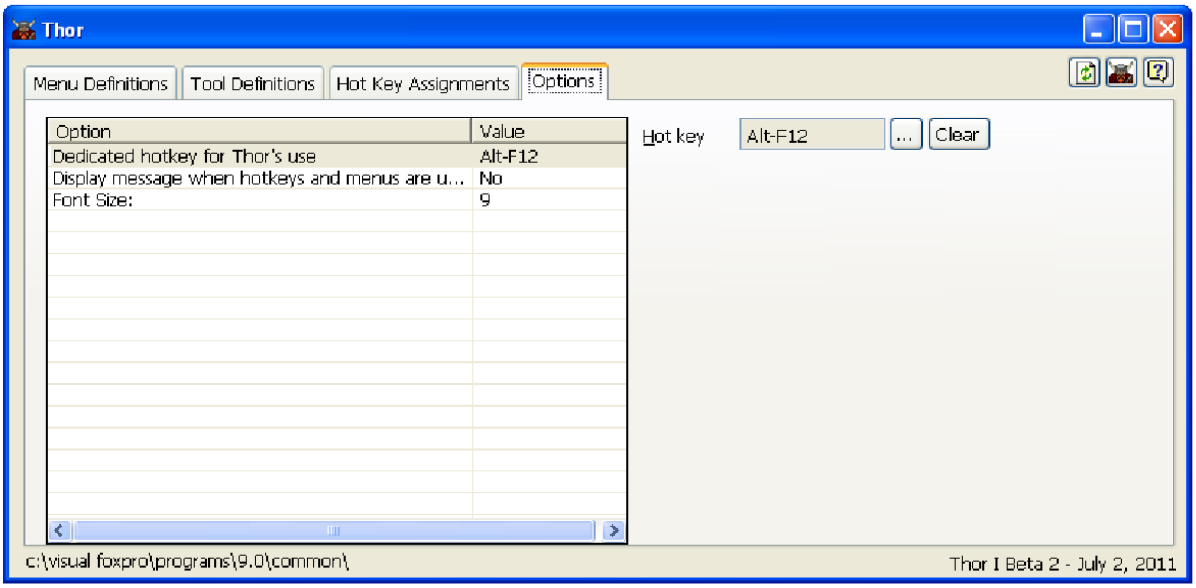
Section 2.5.8. Options

The fourth page (Options) of the Thor form provides for a few option settings. To change a setting, click on the row on the left, and the appropriate controls will appear on the right.

The first option requires some explanation.

The mechanism used by Thor to provide hot keys that use two or more shift keys (Shift, Ctrl, and Alt) requires the use of one dedicated key that is created by On Key Label. The default for this key is Alt+F12.

The first option allows you to change that hot key. Be aware that it must be a key that can be defined by On Key Label, so it can use only one of Shift, Ctrl, and Alt. Furthermore, the On Key Label definition of the previous hot key will not have been removed, so it is recommended that you close your VFP session immediately after changing the dedicated hot key.



Section 3. Tools from PEM Editor w/IDE Tools

PEM Editor has been around for a few years now, and has a reputation as a replacement for five native FoxPro tools: Add Property, Add Method, Edit Property/Method, Property Window, and Document View. The development of the internal structures necessary to support this effort lent itself naturally to other, similar tools. Over time, the number of such tools grew, and they also migrated farther and farther from PEM Editor's origins.

Eventually, it became clear that there were a lot of very valuable tools being built, but because they were coupled with PEM Editor, they tended to get lost behind what first made PEM Editor famous – replacement for native FoxPro tools. The fact that there were numerous completely new tools, unlike anything previously seen in FoxPro, got lost. This was the original impetus for the creation of Thor.

The list of these tools is quite long (as you can see from the table below). Except for the few that require not documentation other than their title, each is described in the following pages.

Registration of the tools from PEM Editor occurs automatically (if Thor is running) when you install PEM Editor by executing:

```
Do PEMEditor.APP
```

Download the current version of PEM Editor from here:

http://vfpx.codeplex.com/releases/view/67528

Note: Some of these tools can be customized using plug-in PRGS, described in section 3.6

Tool	Description
Parent and Super Classes	
Modify SuperClass	Dialog to find all superclasses for an object and to navigate between them
Re-Define Parent Class	Dialog to re-define the parent class of an object.
Compare with Parent Class	Dialog to compare the properties and methods of an object with those in the parent class

Objects, Properties, Methods, and Events	
Copy properties and methods	Copies properties and method code from the current object.
Compare with copied object	Compares properties and method code of the current object with those from the copied object.
Paste properties and method code	Pastees properties and method code from the copied object into the current object.
Paste Object	Pastes (create) the copied object.
Add Object	Dialog for adding an object, using PEM Editor's 'Open Files' dialog.
Find Objects.	Dialog for searching for objects based on the values of any of their properties.
Object Size and Position	Opens the 'Size and Position' form for changing the size and position of objects (Extension of ForPro "Format" menu)
Auto-Rename	Renames current object (and optionally all its child objects) based on the standard 3-letter prefix. *(Plug-in)*
Opening Files, Favorites, and MRUs	
Open Class	Dialog to find and open classes based on part of class name, base class, etc. Can also drag/drop classes onto forms.
Open Form	Dialog to find and open forms based on part of form name
Open Project	Modify Project
Open Other File	Dialog to find and open files based on their extension or part of the file name
Favorites	Manages list of favorite class libraries, classes, and forms.
Open MRUs	Pop-up menu to access MRU s for class libraries, classes, forms, etc.
Code Windows (PRGs, and VCX/SCX)	
Back	Moves back to the previous code window.
Forward	Moves forward to the next code window.

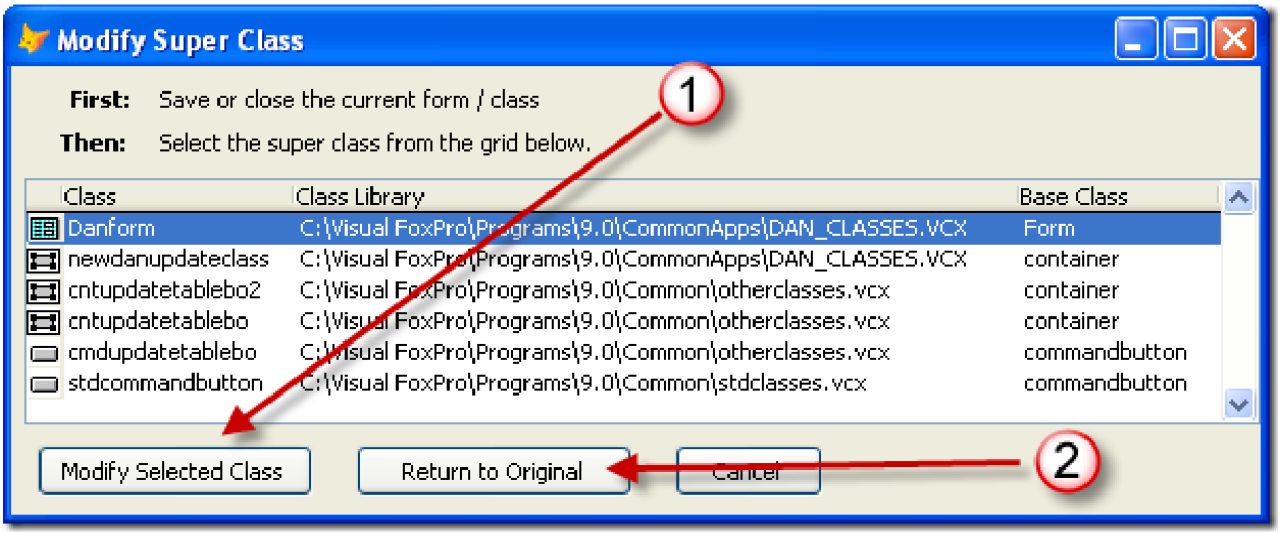
Close all code windows	Close them all.
Close all VCX/SCX code windows	Close all VCX / SCX code windows
Close all code windows but this one	Close all code windows but this one
All Open Windows	Pop-up listing of all open windows (including Form and Class Designers)
Move / Resize Window	Move and resize the current edit window to a specified size and position.
Analyzing / Modifying Method Code	
Go To Definition	Go to the definition of the name under the cursor; method, PRG, procedure, etc. *(Plug-in)*
Create Locals	Create the LOCALs list in a method, based on assignments in that method. *(Plug-in)*
BeautifyX	Beautify to the extreme; highly customizable standardization of method code, with special emphasis on SQL code
Extract to Method	Extracts the currently highlighted block of code into a new method.
Enhanced Cut / Copy	Enhanced Cut / Copy
Control Structures	Highlight current control structure (IF /ENDIF, DO CASE, etc); also close by pasting ENDIF, ENDCASE, as appropriate. *(Plug-in)*
Code Listings	Code listings for all method code; may include inherited code or code from all child objects
Cross References	Categorization of names referenced in code into a dozen categories. The results are displayed in a grid similar to Code References.
Dynamic Snippets	Insert snippets of code that are dynamically created based on parameters. (Extension of Intellisense Auto-completion, which inserts static snippets).

Section 3.1. Parent and Super Classes

Section 3.1.1. Modify Super Class

This tool assists you in opening the parent class (or super class) of an object for editing.

You are presented with a form such as is shown below, which shows the entire parentage of the object, including classes and super classes:



As the directions on the form indicate, you must then:

- Save or close the form or class you are working on (since you cannot open a parent class if you didn't)
- And select the parent class (double-clicking will work.)

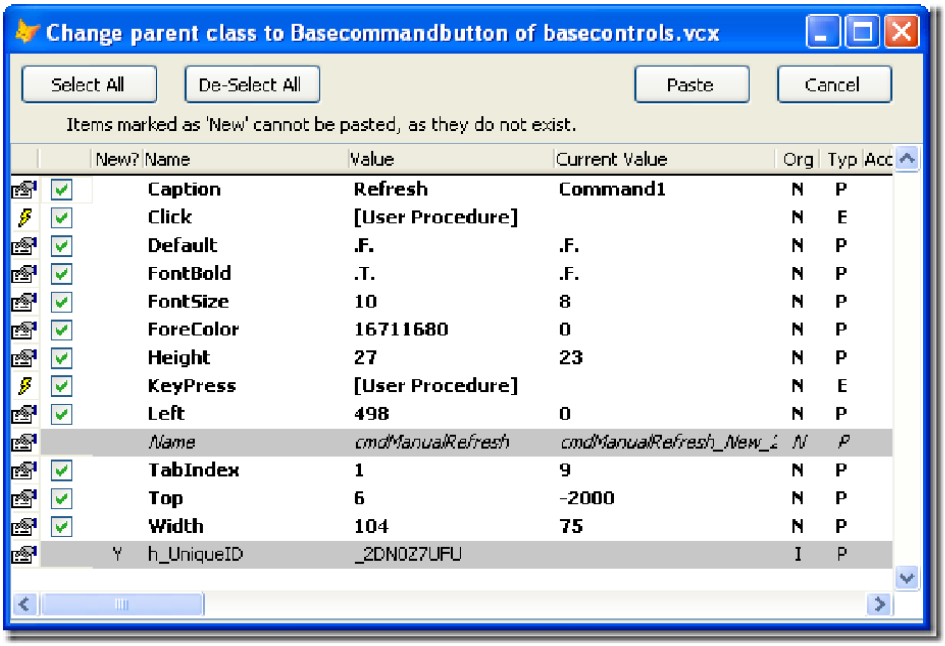
Note that this form remains open after you have selected a class or super class. This allows you to continue to navigate between any of the classes as well as to return to the original class or form you were editing.

Section 3.1.2. Re-Define Parent Class

This feature allows you to re-define the parent class of an object.

You are first prompted for the new parent class for the object (see note at the bottom). There will be an additional warning prompt if you attempt to change the baseclass of the object.

You are then presented with a form like this, where you complete the process of re-defining the parent class:

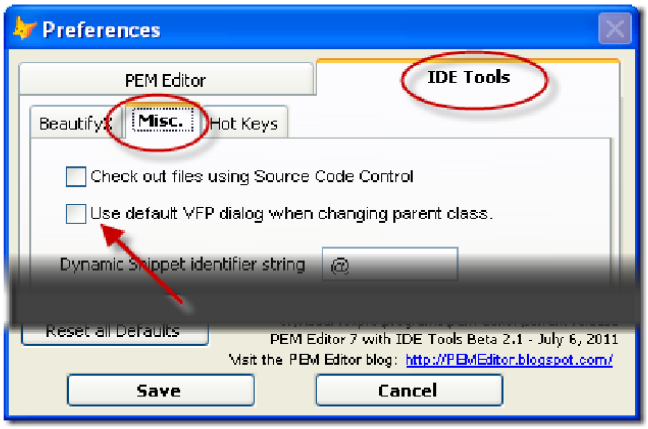


This form shows you which properties or methods cannot be copied because they do not exist in the parent class (greyed out), allows you to select which of the remainder you wish to have copied (normally all - but not always), and is a last-minute check before actually re-defining the parent class.

Note that changing the parent class does not work for either:

- Container objects (containers, grids, etc.) which have any child objects.
- Objects which are members of a parent class (and thus cannot be deleted).

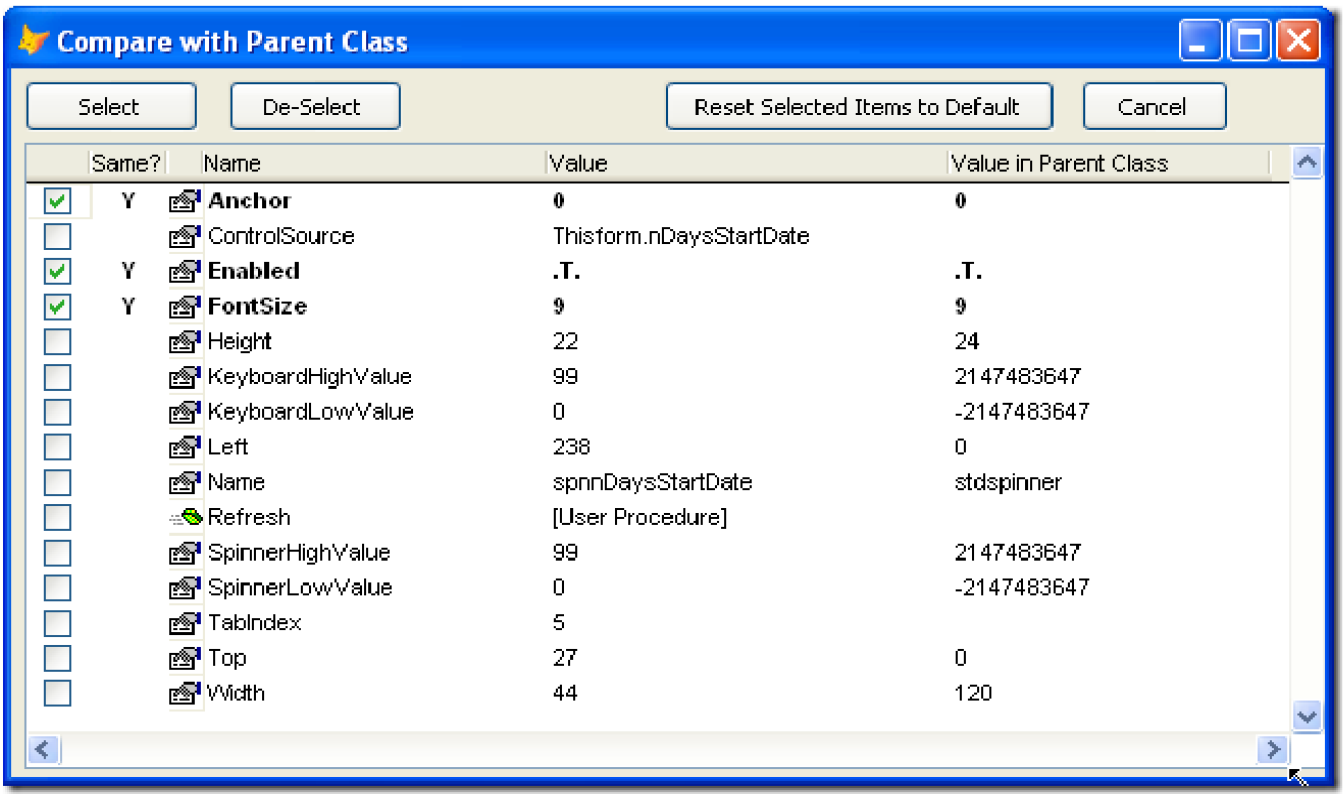
The default form used to find the new parent class is the 'Open Class' form from IDE Tools. This can be modified to use the default VFP 'Modify Class' dialog by using the Preferences form.



Section 3.1.3. Compare with Parent Class

This tool displays a new form which shows all non-default properties and methods for an object, along with their current values and the values that would have been inherited from the parent class. PEMS that are identical to their inherited values are highlighted and may be reset to default.

Note that if the object is a member of a parent class, it actually digs into the parent class that the object is a member of, and extracts all non-default properties and methods from there as well.



All values which are identical to the parent class are initially checked off (and bold). The list of selected items can be modified as needed. Then, if desired, the button 'Reset Selected Items to Default' can be used.

This feature is also used as the final step when copying properties and method code from an object into its parent class during re-factoring.

Section 3.2. Objects, Properties, Events, and Methods

Section 3.2.1. Copy properties and methods

This feature copies properties and methods from the currently selected object into a "clipboard" for later pasting or comparing.

The copied properties and methods can be used a number of different ways:

- By selecting a different object and then using Compare with copied object
- By selecting a different object and then using Paste properties and method code
- By using Paste Object to create a new object.

The "different object" can be another object on the same form or class, in a different form or class, or in the parent class.

This feature is also used as a final step when copying properties and method code from an object into its parent class during re-factoring. See Paste properties and method code.

A few notes of interest:

- This tool allows you to copy any object, even if VFP cannot copy it (the familiar message "Cannot copy objects because some are members of a parent class"). In this case, the tool actually digs into the parent class that the object is a member of, and extracts all non-default properties and methods from there as well.
- You can copy an entire class when editing to simplify later pasting:
    - Use this tool on the class
    - Close the class.
    - Open the form or class to paste into.
    - Use Paste Object.
- The "clipboard" that the properties and method code gets pasted into survives even if you close PEM Editor; they will survive as long as 'Clear All' is not used, which removes the object containing them from memory.
- Unlike VFP Copy, this tool can only copy a single object.

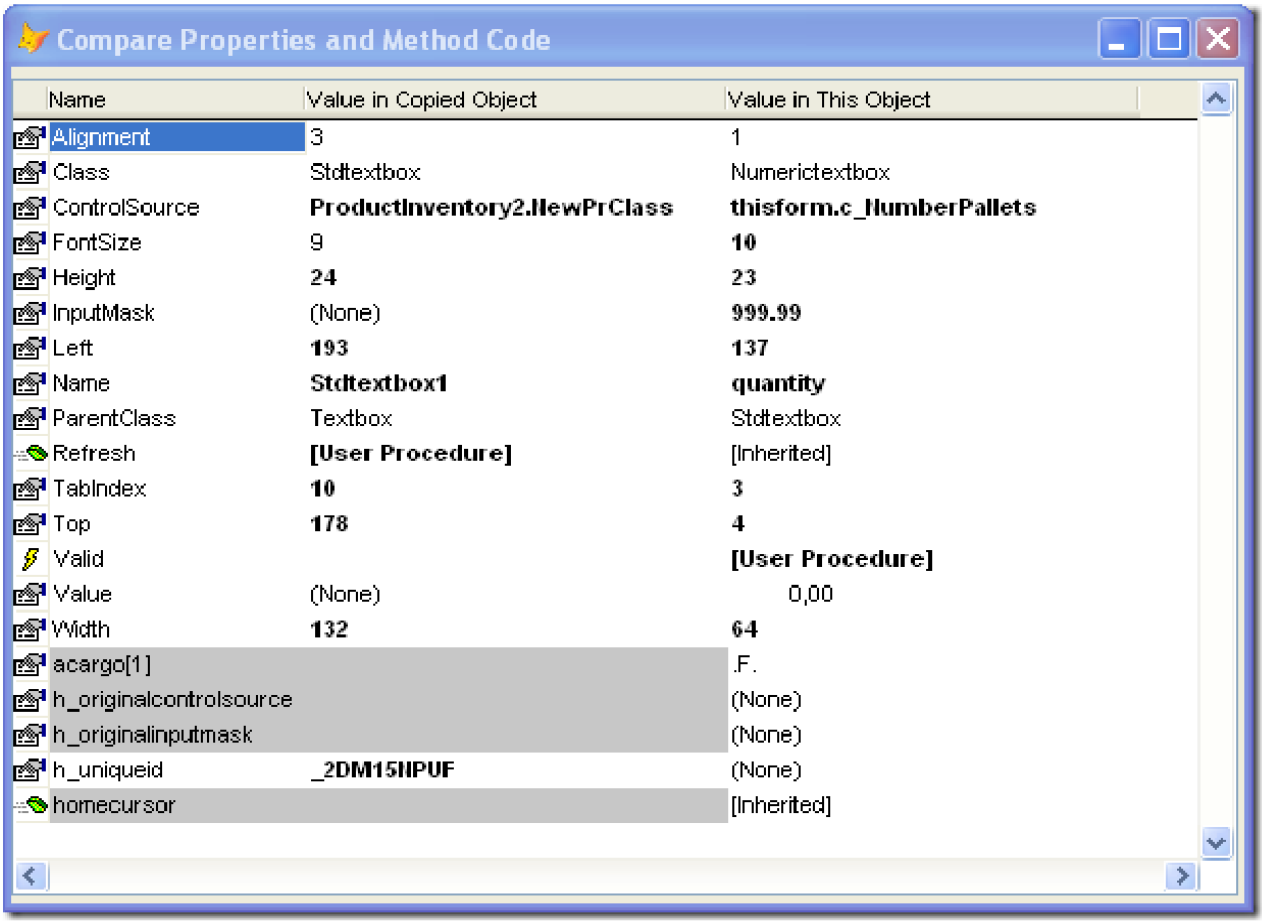
Section 3.2.2. Compare with copied object

This tool compares all the properties and methods in the copied object (see Copy properties and method code) with those in the current object.

The listing shows all non-identical properties or methods.

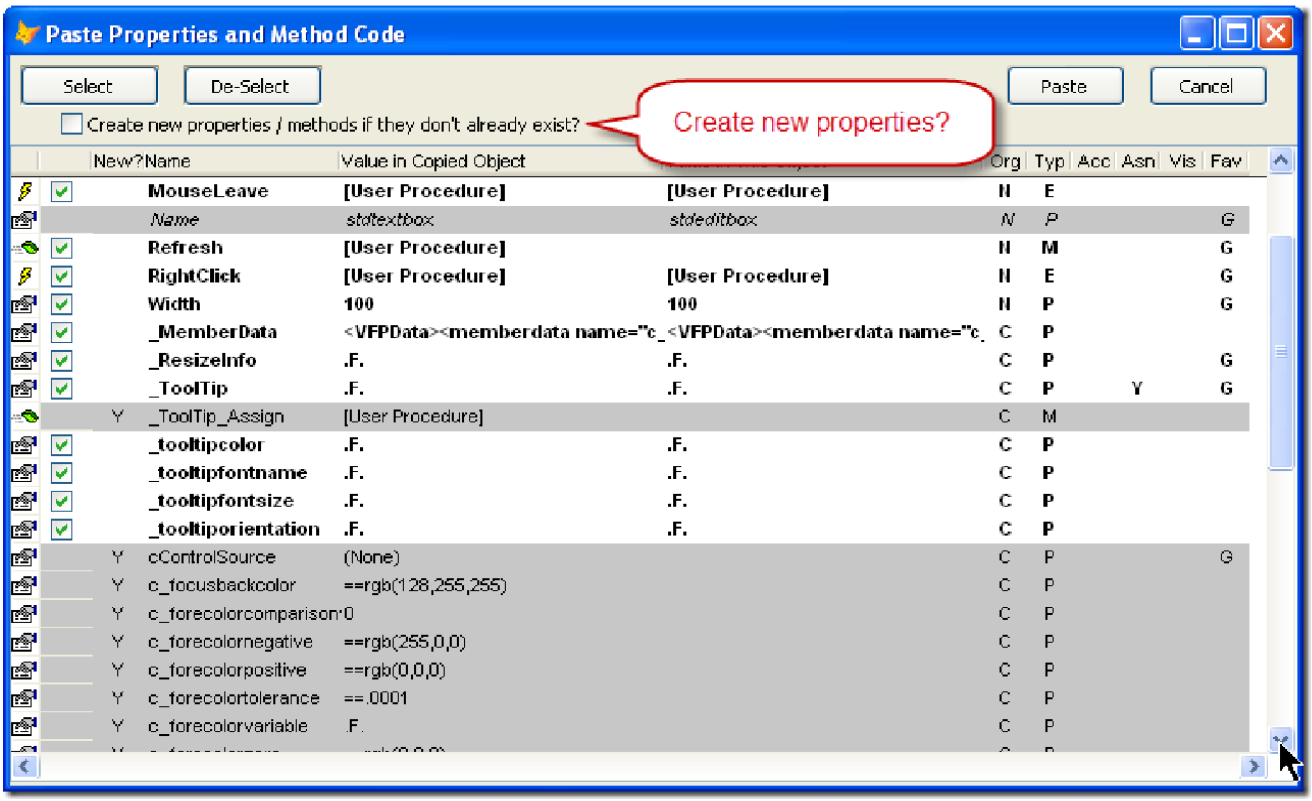
- Non-default properties or methods are shown in grey.
- Missing properties or methods appear with a grey background.

The following shows a comparison between two different objects of the same baseclass, but differing classes.



Section 3.2.3. Paste properties and method code

This tool pastes some or all of the properties and method code that has been copied (see Copy properties and method code) into another object. The object being copied into can be in the same form or class, in a different form or class, or into the parent class (more on this last point at the end of this section).



This form shows you which properties or methods cannot be copied because they do not exist in the object (greyed out) and allows you to select which of the remainder you wish to paste.

If pasting into a form or class (rather than into an object contained in a form or class), it is possible to create new properties and events. In this case, there will be a checkbox where you can indicate whether you want to create new properties or methods if they don't already exist.

As mentioned in the elsewhere, the "clipboard" that the properties and method code gets pasted into survives even if you close PEM Editor; they will survive as long as 'Clear All' is not used, which removes the object containing them from memory.

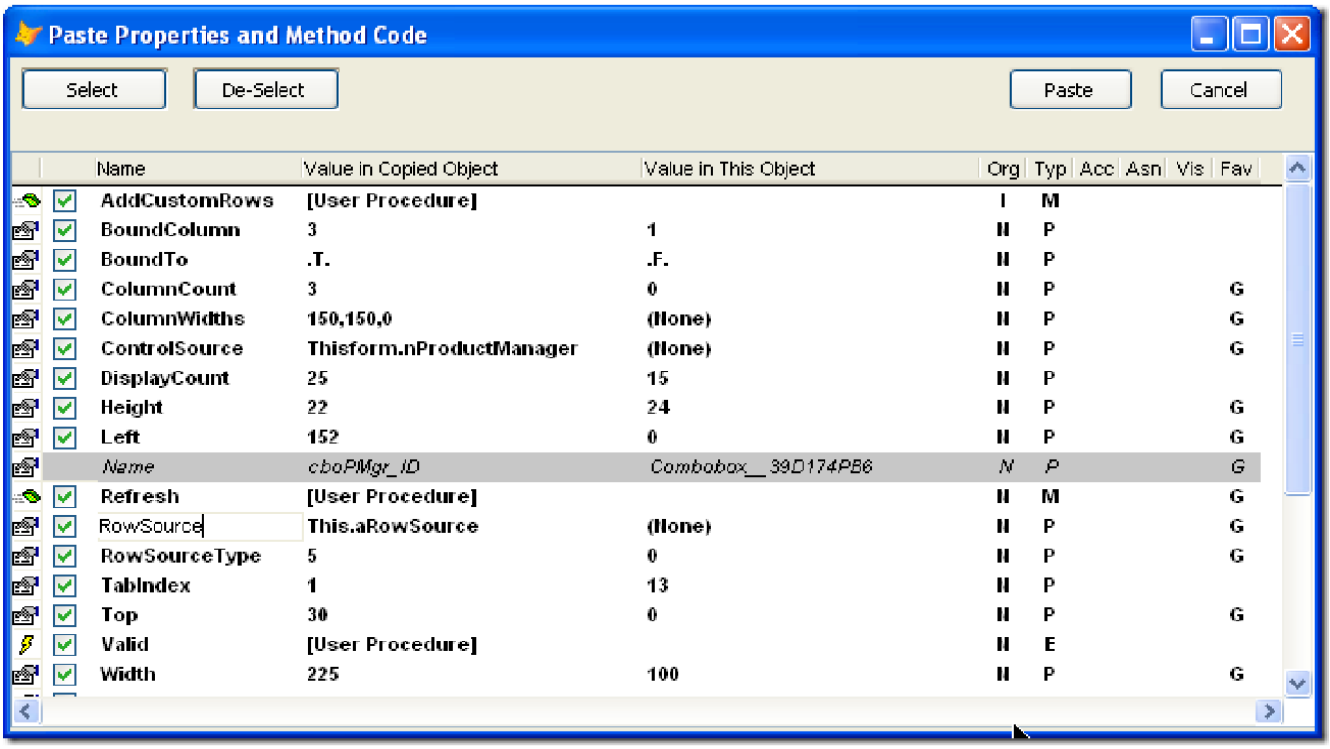
When **re-factoring**, the following sequence can be used to copy properties and methods to the parent class of an object:

- Use Copy properties and method code
- Use Edit Parent Class
- Use this tool to paste the desired properties/methods into the parent class.
- Close the parent form/class and re-open the original form/class.
- Use Compare with Parent Class to reset to default any properties or methods which are now identical to the properties or methods in the parent class.

Section 3.2.4. Paste Object

This feature creates a new object, using the class name and library captured by "Copy Object" and then pastes the properties and method code that has been copied into that object.

The new object will be a child of the currently viewed object, if that object can have children; otherwise, it will be a sibling of the currently viewed object.



This form shows you allows you to select which of the PEMs from the source object you wish to paste into the new object.

Paste Object works essentially the same as VFP's familiar Paste of an object, with the following enhancements:

- You can copy any object, including those which VFP cannot (the familiar message "Cannot copy objects because some are members of a parent class")
- You can copy an entire class when editing it. Then, when the class is closed, it can be paste into another form or class.
- You can select which properties, methods, and events you wish to paste.

As mentioned elsewhere, the "clipboard" that the properties and method code gets pasted into survives even if you close PEM Editor; they will survive as long as 'Clear All' is not used, which removes the object containing them from memory.

Section 3.2.5. Add Object

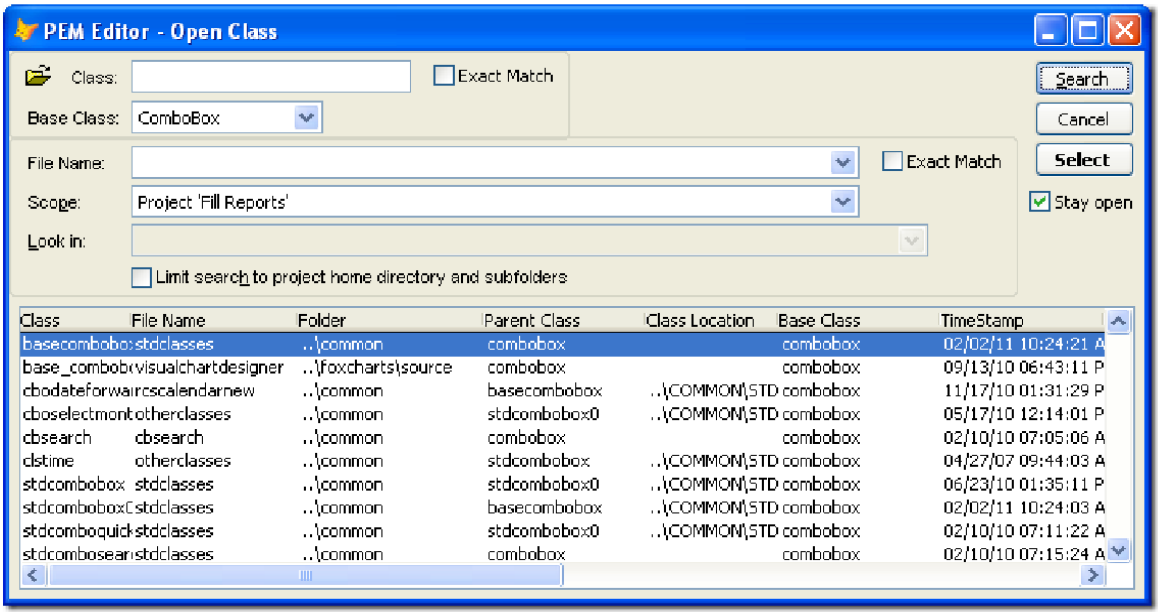
This tool adds a new object to the currently selected object.

The dialog for selecting the object to add (see below) allows searching in a number of different ways:

By project or folder

- By some of all of the class name
- By the class's base class

This was originally implemented in response to a user request to make it easier to add objects into columns.

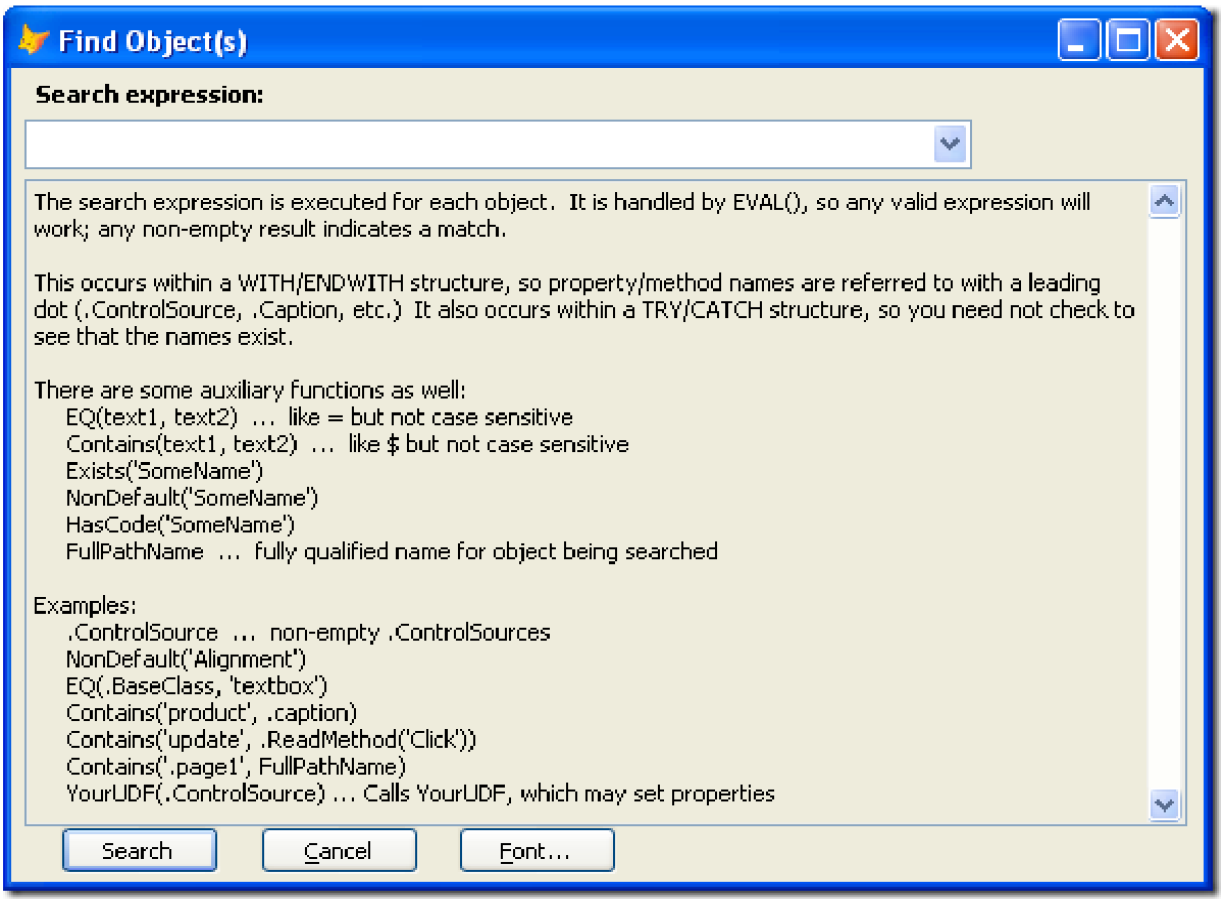


Section 3.2.6. Find Objects

Find Objects provides the ability to search through all objects in a form or class based on values of their properties, to display the list of found objects in a grid for easy review, and then to modify the values of properties for those objects, either individually or all at one time.

Step 1: Using the 'Find Object(s)' search form

This form looks like this:



The search expression entered here will be executed for each object in the form or class. If the expression returns a non-empty result, that object will be considered a match.

The search expression is executed within a WITH / ENDWITH structure for each object, so that properties can be referenced simply: .ControlSource or .Caption, for instance.

The search expression is also executed within a TRY / CATCH structure, so that there is no need to be concerned about non-existent properties.

Many searches are quite simple then: ***.ControlSource*** will find all objects with non-default controlsources.

The search expression can, however, be as complicated as desired, and can even call your own UDFs. (More on this in a bit.)

In addition, there are some auxiliary procedures and functions available:

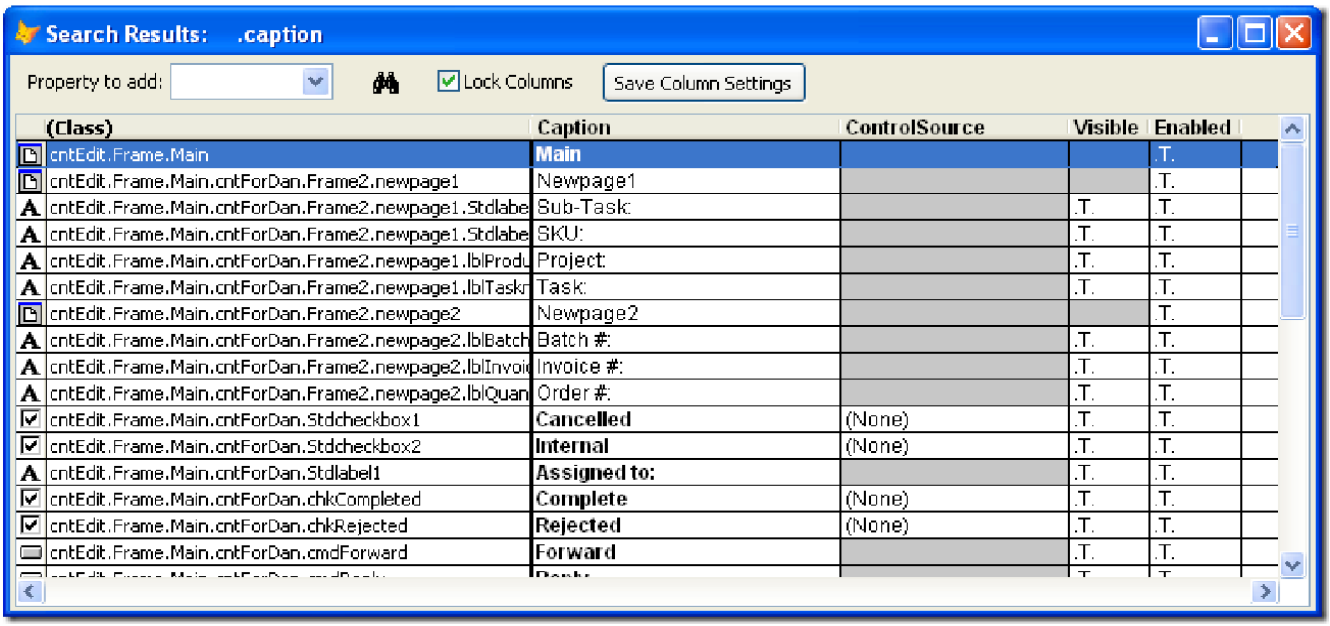
EQ(text1, text2)	Like =, but not case sensitive
Contains(text1, text2)	Like $, but not case sensitive
Exists('SomeName')	Is "SomeName" a property, method, or event?
NonDefault('SomeName')	Does "SomeName" exist and is it non-default?
HasCode('SomeMethod")	Does "SomeMethod" exist and does it have code, either here or inherited?
FullPathName	Fully qualified name for object being searched

Some examples of usage:

.ControlSource	non-empty .ControlSources
EQ(.BaseClass, 'textbox')	for those of us who cannot remember case of baseclasses
Contains('product', .caption)	'product' appears anywhere in the caption
Contains('update', .ReadMethod('Click'))	'update' appears in the Click method
Contains('.page1', FullPathName)	All objects in '.page.'
YourUDF(.ControlSource) ...	Calls YourUDF, which may set properties

Step 2: The Results Grid

The results grid has one line for each objects, and columns which show the name of the object and various other properties:



Any properties referenced in the search expression are automatically shown as columns in the grid. You can also control the grid display in a number of other ways:

You can add properties from the combobox at the top left.

You can also add properties by using Ctrl-Click on a row in the grid in the PEM Editor form (see the next section).

You can remove properties by using the right-click combobox on the column,

You can move or resize the columns.

You can save the column settings so that the grid will be displayed with the same columns, etc, the next time.

You can change the value of any property by double-clicking on it, then using the popup property editor. You can also reset values to default by using the right-click context menu. (See also the next section for changing all the values for a property at once.)

You can also select one of the objects to be displayed in the PEM Editor form by double-clicking on the object name in the first column. When possible, this will also cause the object to be selected in the form or class being edited.

Co-ordination with PEM Editor

If the PEM Editor form is open when a search is done, it will show the list of all properties and methods, just as if you had selected a number of objects on the form or class being edited using the mouse and /or keyboard. This allows you to change the value of a property for all selected objects all at once, either by double-clicking on the property, or by resetting the property to default (in the right-click context menu).

You can also cause properties to be added to the Search Results form by Ctrl-Clicking on them in the PEM Editor form.

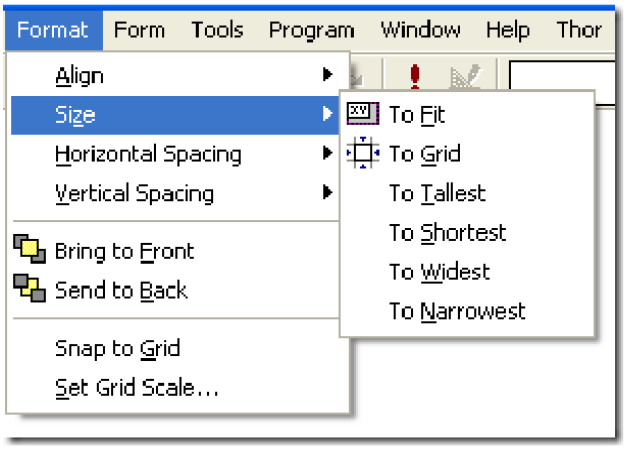
Finally – When you have selected a number of objects in the form or class you are editing, you can actually use the 'Search Results' form to edit the values for single properties individually. Even if the 'Search Results' form is not open, ctrl-clicking on a property when you have selected multiple objects will open up the 'Search Results' form and add the property as a column there, where the values can be reviewed and edited.

Section 3.2.7. Object Size and Position

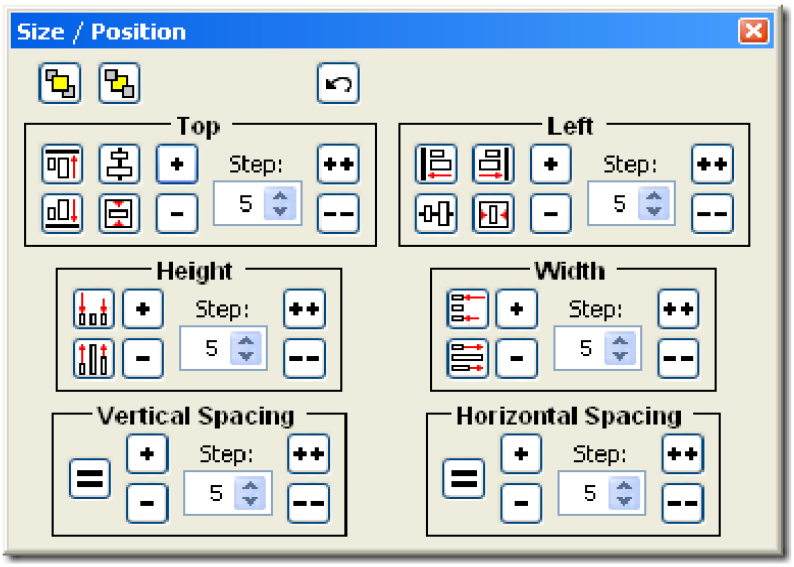
The 'Object Size and Position' form combines almost all of the features of the native VFP Format Menu (below, on the left) into one form (below, on the right).

This was done so that all the features from the Format Menu could be available with one click, and to simplify access to those which may be used multiple times consecutively (such as when increasing a dimension by a pixel).

VFP Native Format Menu



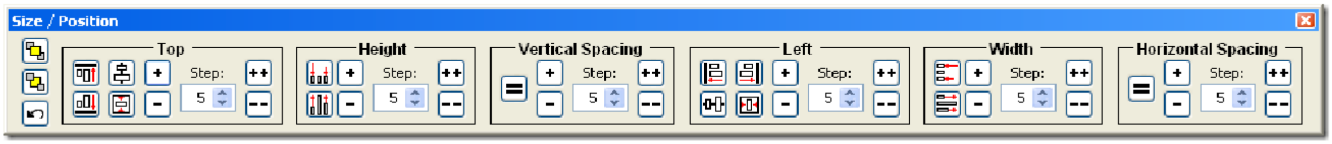
Object Size and Position Form

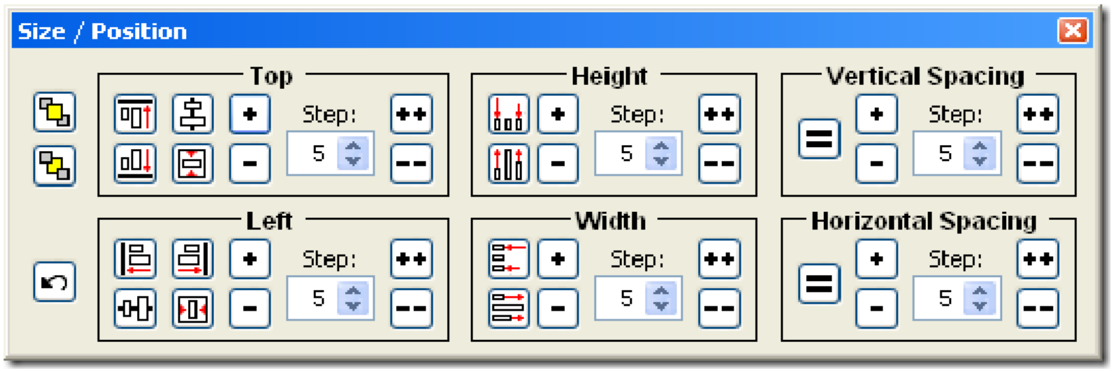


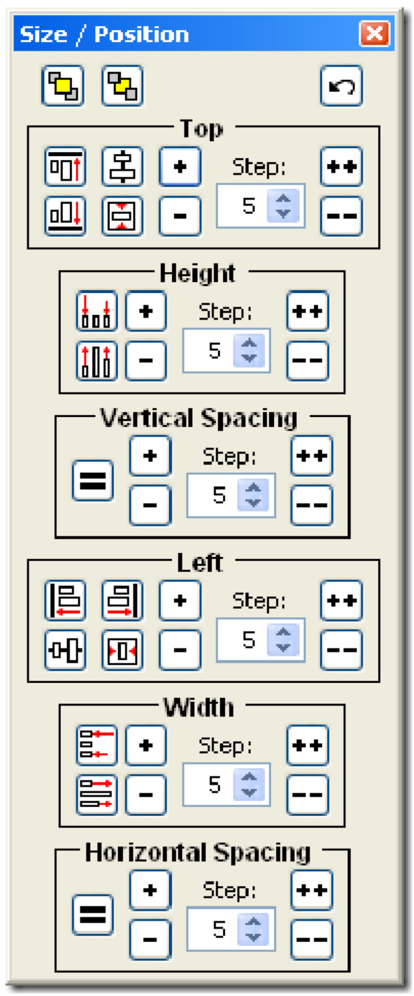
Notes:

- Not <u>all</u> of the features of the Format menu have been duplicated here. The following are missing:
- Size to fit
- Size to grid
- Snap to grid
- Set grid scale
- There are some new features. For each of the dimensions and spacings, you can increment or decrement either by a single pixel at a time or you can select a larger step size.
- There is also an UnDo button, which will reverse any actions taken since the form took focus.

- The containers within the form get moved around as the width of the form changes. This is provided so that the form can be docked either vertically or horizontally. Below are the displays for each of the other configurations:







Section 3.2.8. Auto-rename

Applies to tools:

- Auto-rename current object
- Auto-rename current object and all its child objects

In the VFP Help File, the section entitled 'Object Naming Conventions' provides a recommended format for naming objects - a three letter prefix, based on the type of object, following by some meaningful name. Curiously, when adding new objects to a form or class, VFP does not use this convention by using the suggested three letter prefix.

This tool provides the capability to rename objects so that they use this naming convention (that is, to rename all those objects that still have the default name created for them by VFP). The name created will have the recommended prefix, followed by all or part of the Caption, ControlSource, or RecordSource (if they exist), or else a number; in any case, the new names will be uniquely assigned.

Notes:

- For tool *Auto-rename current object and all its child objects*, objects are only renamed if they still have the default name created for them by VFP. (There is a pop-up that notifies you of any objects that were not renamed).
- This option does not replace references to objects in any method code. Any such changes would have to be made manually. Thus, it is recommended that this option be used very early, before there are any references to the objects in method code.

Customization: There is a plug-In PRG that allows you to modify the behavior of this tool to fit your own needs. The plug-in is called with the suggested new name, which can be modified if desired. See section 3.6, "Plug-Ins".

Section 3.3. Opening Files, Favorites, and MRUs

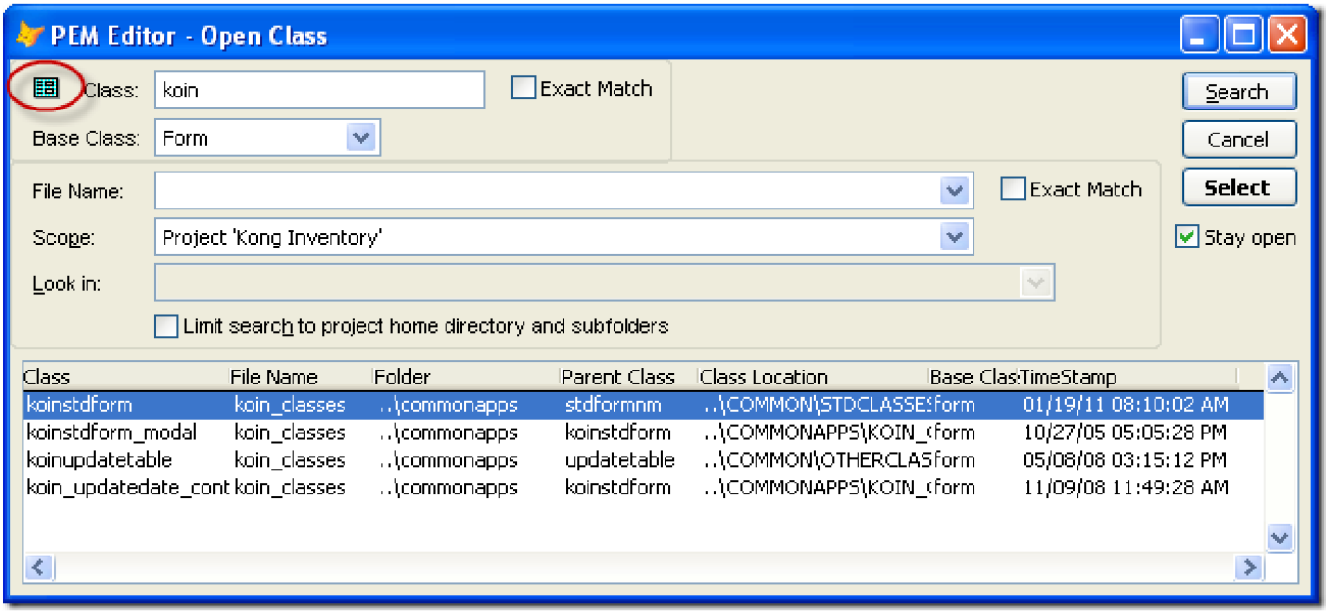
Section 3.3.1. Open Class, Form, Project, or Other

These dialogs are very similar, allowing searching by some or part of the file name, as well as within the current project or the current folder.

In all of the dialogs, you may move or resize the columns, as well as clicking on the column header to sort.

The dialog for opening a class has three additional features:

- You can search on some of or all of the class name (partial match to 'KOIN' here).
- You can search by the base class you are looking for.
- You can select a row from the grid and then drag the icon in the upper left corner to drop it
- onto a form or class OR
- onto the PEM Editor form, which will cause it to be added to whatever object is currently displayed in the PEM Editor grid. Normally not necessary, this is quite handy for those times when dragging and dropping is cumbersome, such as when adding controls to a column or to a container that is obscured by other objects.



Section 3.3.2. Favorites

There are three different groups of favorites that you can maintain:

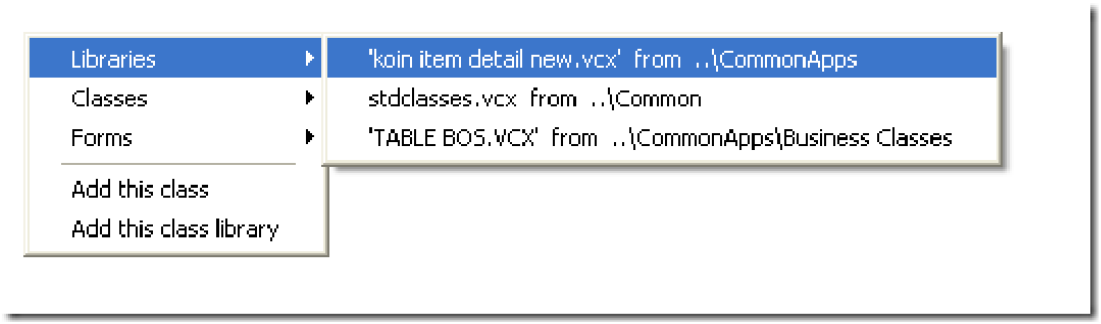
- Class Libraries
- Classes
- Forms

When you are editing a class, you can add that class (and/or class library), as you can see in the example below. If the class or class library is already in the Favorites list, there will be options available to remove it from the Favorites list.

Similarly, if you are editing a form, you can add the form to the list, or remove it if it already in the list.

Notes:

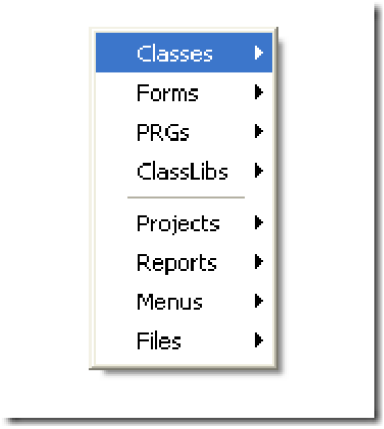
- There is no limit to the number of entries. This does not guarantee, however, that the popup menu will be able to display all of them!
- By default, selecting a class library from the list will open up the class browser.



Section 3.3.3. Open MRUs

This tool pops up a menu of eight eight different MRU lists, as you can see here.

MRUs are the forgotten step-child of FoxPro, but this tool makes them more accessible, and other tools have dramatically increased the number of places that keep the MRU lists up to date. See also section 3.7, "MRU lists and Source Code Control.".



These lists are the same as are created and used in the Command Window, and are stored in your resource file.

Note the addition of one new list, not maintained by FoxPro: Class Libraries.

Section 3.4. ***Windows Code Windows (PRGs, and VCX/SCX)***

Unless otherwise stated, the "code windows" referred to here are all code windows for PRGs and method windows for SCXs and VCXs.

Section 3.4.1. Back (previous method)

Cycles backward through all code windows.

Section 3.4.2. Forward (next method)

Cycles forward through all code windows

Section 3.4.3. Close all code windows

Closes all code windows, attempting to save any changes. That is, as if you press Ctrl+W for each window. Thus, this will be interrupted if there are any compilation errors in any of the windows.

Section 3.4.4. Close all VCX/SCX code windows

Same as "close all code windows", above, but applies only to windows from SCXs and VCXs.

Section 3.4.5. Close all code windows but this one

I'm not going to explain this one. Make a guess

Section 3.4.6. Pop-up of all open windows

Pops up a menu listing all open windows. This is the same list as can be generated from the Window pad in the VFP system menu, except that the list is ordered by type of window and is much more readable when there is a longer list of windows (i.e., more than nine).

Section 3.4.7. IDE Tools: Move/Resize Window

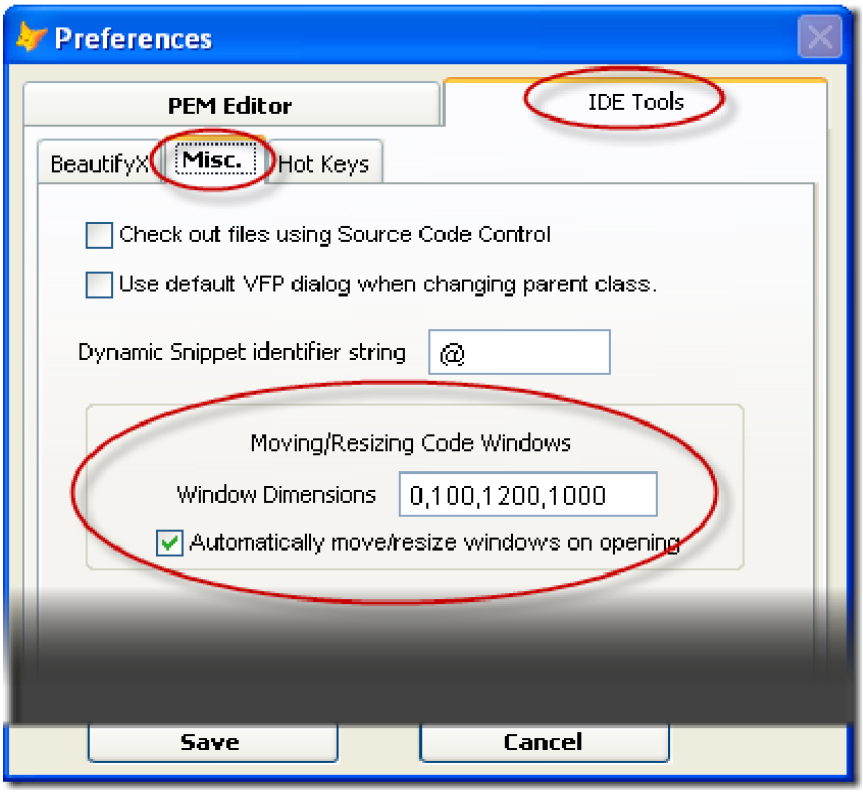
This tool allows you to move and/or resize the currently open code window.

The dimensions to be used for the new window need to be defined in the Preferences form.

They are entered, in pixels, in the following order: Left, Top, Width, and Height.

Notes:

- You may skip any dimensions that you want to ignore.
- The window will not be resized so that it is too wide or too tall; thus, you can set the Width and Height values to be very large (say, 5000) and the window will expand to the maximum that will fit.
- The check box about automatically doing this on opening windows applies to all windows opened by PEM Editor, Document TreeView, or and of the IDE Tools that come with PEM Editor.



Section 3.5. Analyzing / Modifying Method Code

Section 3.5.1. Go To Definition

Go To Definition, modeled after "View Definition" in Visual Studio, allows you to point to a name that is referenced in your code, and go to (that is, display and/or edit) its definition. It can be used to create new methods and properties in a form or class.

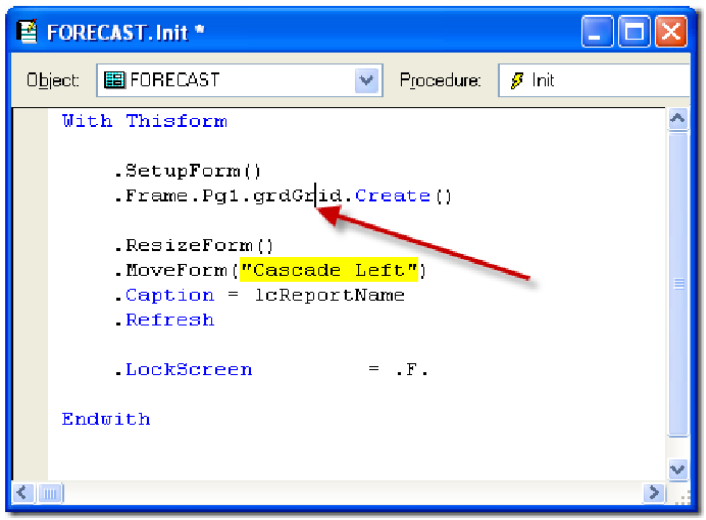
The mechanics are simple:

- Click on the name.
- Run Go To Definition.

The "name" that is searched for is determined as follows:

- All characters to the right of the cursor (or of the selected text, if any) that can be part of a name (letters, numbers, and underscore) are included.
- All characters to the left of the cursor (or of the selected text, if any) that can be part of a name, including periods, are included.
- If the leftmost character is a period, and is only preceded by whitespace, then the preceding code is examined looking for WITH statements. This is done repeatedly, handling embedded WITHs.

In the following example, the name being searched for is "Thisform.Frame.Pg1.grdGrid". (Note that the cursor does not have to be within the word; in this case it could have been after the period on the left or before the period on the right.)



3.5.1.1. Searching in the current form / class

The following table summarizes all the different types of names that can be searched for in the form or class currently being edited.

Note that it is not always possible to resolve what 'THIS' refers to when looking at method code. It is possible to learn the name of the object that owns the code, but not what the full path name of the object is. Thus, it is possible, if there are objects having the same name, that the search will find the wrong object. This is alleviated somewhat if the method code was opened by PEM Editor or Document TreeView, as they change the title of the window to display the full name of the object.

Type of Name:	Action taken:
A method	Opens up the method for editing, if there is any local non-default code; otherwise opens up a txt file containing all the inherited code for the method.
An object	Selects that object, if possible, for display in PEM Editor and property window. There are some conditions where this fails: If the object is hidden beyond other objects or not otherwise visible, and if it is on a different page of a pageframe (this list seems to fail intermittently).
A property	Same as selecting an object, but also tries to select that property in the PEM Editor grid. This only works if the PEM Editor form is open, and may also fail based on the filters in effect in the grid.
DODEFAULT	Opens up a txt file containing all the inherited code for the method.
THIS	Same as 'an object', above.

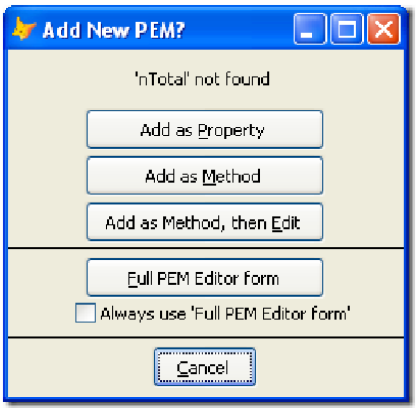
3.5.1.2. Searching in other files.

If the search is to be conducted looking in file(s) other than the form or class being edited, the files in the active project, if any are searched; if there is no active project, then all files in the path are searched.

Type of Name:	Action taken:
a PRG	Opens the PRG.
a PROC or FUNC in a PRG	Opens the PRG and highlights the start of the Procedure or Function
a Constant (#Define …)	Opens the #Include file, and highlights the constant
a Form	Opens the form
a Class	Opens the class, whether it is in a VCX or PRG
CREATEOBJECT or NEWOBJECT	Opens the class, whether it is in a VCX or PRG

3.5.1.3. Creating a new Property or Method

If the name being searched for looks like it could be a new Property or Method, then the following form will pop up allowing you to create it on the fly (adding it to _MemberData as appropriate, assigning the mixed case name for the new PEM).



3.5.1.4. Customization

If there is still nothing found, there is this: There is a plug-In PRG that allows you to perform you own search for the name. (For example, the name could be an alias for a table; the table could be opened for browsing.)

See Section 3.6. “Plug-Ins”.

Section 3.5.2. Create LOCALs

Create Locals provides for the creation of LOCALs statements for variables which are assigned values in a method or procedure.

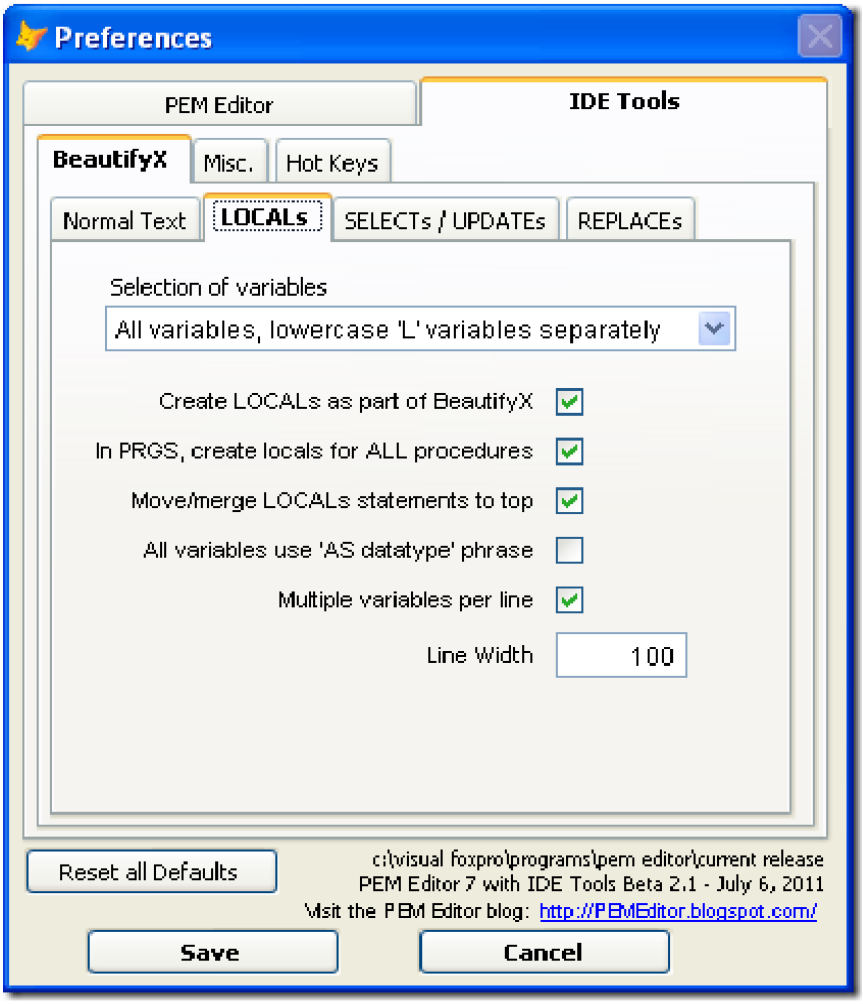
Once the list of variables assigned in the method or procedure is harvested, the LOCALs statements are created. They appear at the top of the method or procedure, before any existing code, with the variables in alphabetical order.

- There are a number of user preferences available (see the preferences form, below):
- Controlling which variables are added to the LOCALs statements, one of:
- Lowercase 'L' variables only
- Lowercase 'L' variables only, with a commented list of others not beginning with lowercase 'L'
- All variables, with lowercase 'L' variables listed separately
- All variables, merged
- For PRGs, controlling whether the tool applies to the entire PRG or just for the one where the cursor is
- Controlling the format of the LOCALs statements:
- All other LOCALs statements can be merged into the newly created statements.
- The implicit datatype for each variable ('lnCounter as Numeric") can be displayed.
- You can control how many variables are displayed per line:
- If you do not check 'Multiple variables per line', you will get one LOCAL statement with many continuation lines, one variable per line.

```
Local lnDay, ;
    lnI, ;
    lnPeriod
```

- If you do check 'Multiple variables per line', you will get multiple LOCAL statements, each with as many variables as will fit within the line width that you supply. Note that if you supply a very small line width, you will actually get one LOCAL statement for each variable.

```
Local lnDay, lnI, lnPeriod
```



3.5.2.1. Customization

There is a plug-In PRG that allows you to modify the behavior of this tool to fit your own needs.

See Section 3.6. “Plug-Ins”.

The plug-in is called with an array of all the harvested variable names, and returns a character string result which will pasted into the method code. Thus, the plug-in can determine which variables will be placed in the LOCALs list and how they are displayed. There is also enough information to identify ‘orphan’ locals – that is, variables that appear in the LOCALs list but are not assigned a value anywhere.

Section 3.5.3. BeautifyX

BeautifyX provides a wide range of customizable features for improving the display and readability of code. It is used when you are editing code, and is applied either to the highlighted text (if any) or to all of the code. The beautification does not apply to comments, character constants, or TEXT / ENDTEXT blocks (although exceptions to this are provided).

There are four different areas covered by **BeautifyX**.

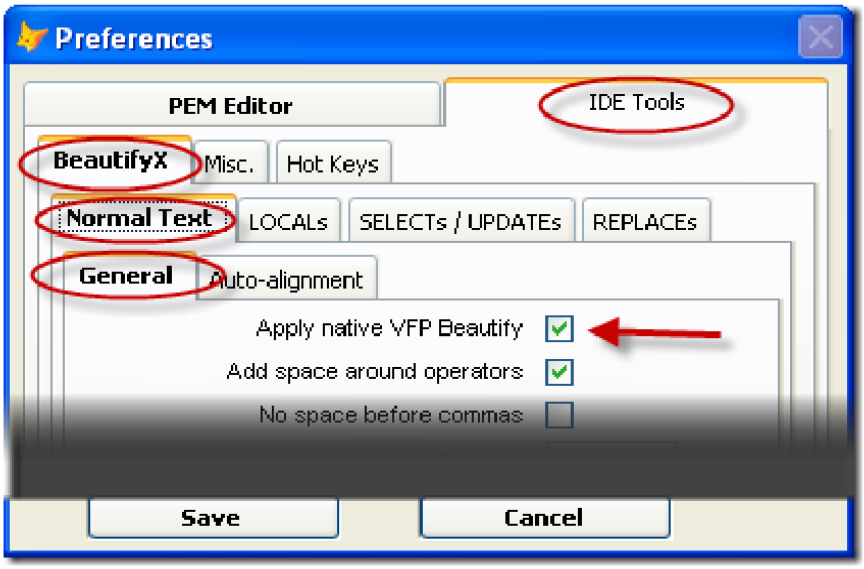
- Native FoxPro Beautify (fixing one bug along the way)
- Creating Locals (see Creating Locals)
- Styling for normal code
- Styling for SQL statements (Select, Update, and Delete) and Replace

All of these features are controlled by the Preferences Form for IDE Tools:

3.5.3.1. Native FoxPro Beautify

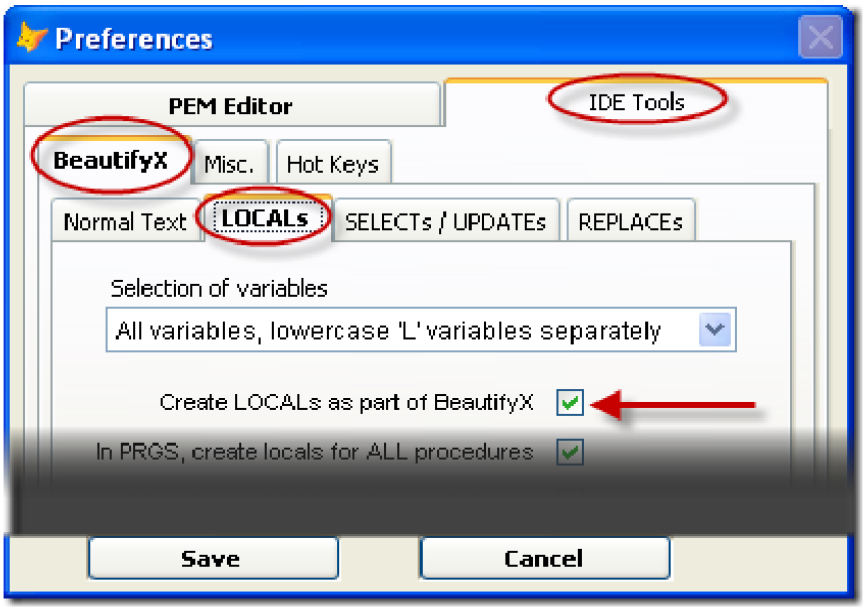
If selected, BeautifyX will first apply native FoxPro Beautify.

There is a bug in native FoxPro Beautify having to do with the handling of mixed case keywords in continuation lines. This bug is corrected by BeautifyX.



3.5.3.2. Creating Locals

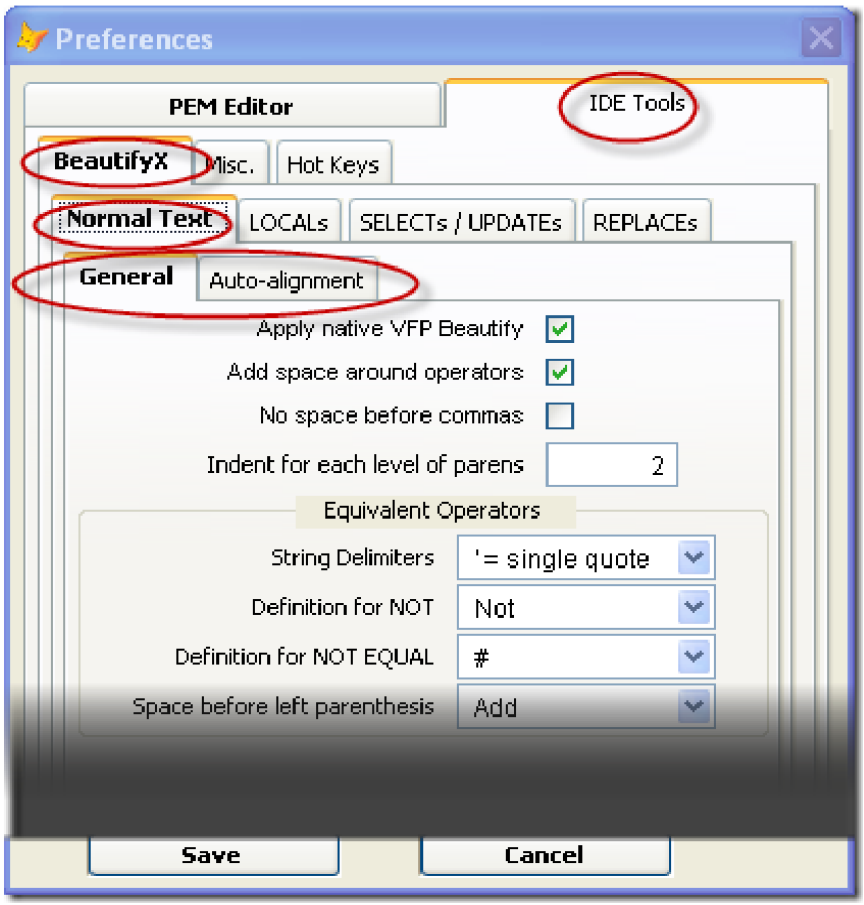
BeautifyX can also incorporate the previous tool, Create Locals, so that it is run at the same time as the rest of BeautifyX.



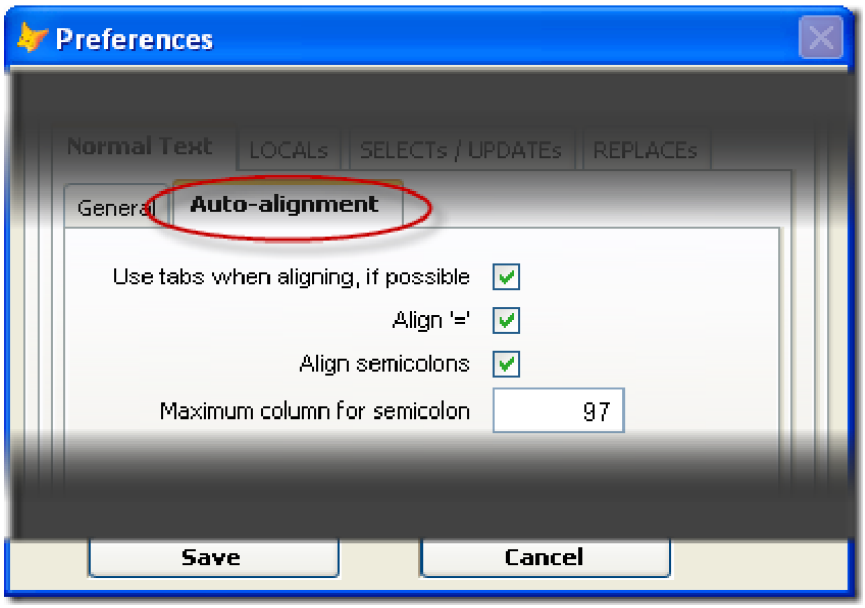
3.5.3.3. Styling for normal code

There are a number of options that control styling of normal code:

- Insuring that there is at last one space on either side of operators (+, –, /, *, =, #, $, %, >, <, !)
- Removing spaces before commas
- Indentation to be added on continuation lines for each level of parentheses
- Preferred operators when FoxPro has equivalent operators:
- single quote, double quote, or brackets
- NOT or !
- #, !=, or <>
- Handling for spaces before left parentheses (add, remove, or leave unchanged.)
-



-

- Alignment across multiple lines:
- For = signs (assignments on consecutive lines)
- For semi-colons marking continuation lines, with a value for the maximum column. Note that you must assign a maximum column here (say, 80), otherwise all the semi-colons are driven as far to the left as possible.
-


3.5.3.4. Styling for SQL statements

Styling for SQL statements (Select, Update, and Delete) takes statements that look like this:

```
Select Detail.*;
    , Cast (Nvl (Dimensions.MCUnits, 1) As N(10)) As MCUnits;
    , Cast (0 As N(10)) As CasesShip;
    From ShippedDetails As Detail;
    Left Join Dimensions;
    On Detail.Partno = Dimensions.Partno;
    Join PartsList;
    On Detail.Partno = PartsList.Partno;
    Where Ordnbr = Thisform.cOrdNbr;
    Into Cursor crsr_Source Readwrite
```

and applies standardized, customizable styling to make them look something like this:

```
Select  Detail.*                                    ;
    , Cast (Nvl (Dimensions.MCUnits, 1) As N(10))     As  MCUnits    ;
    , Cast (0 As N(10))                           As  CasesShip  ;
  From ShippedDetails                               As  Detail     ;
    Left Join Dimensions                                         ;
      On Detail.Partno = Dimensions.Partno                       ;
    Join PartsList                                               ;
      On Detail.Partno = PartsList.Partno                        ;
  Where Ordnbr = Thisform.cOrdNbr                                ;
  Into Cursor crsr_Source Readwrite
```

All of the styling applied to SQL-Select, Update, and Delete statements, as well as Replace statements are controlled on the preferences form (see screenshots below).

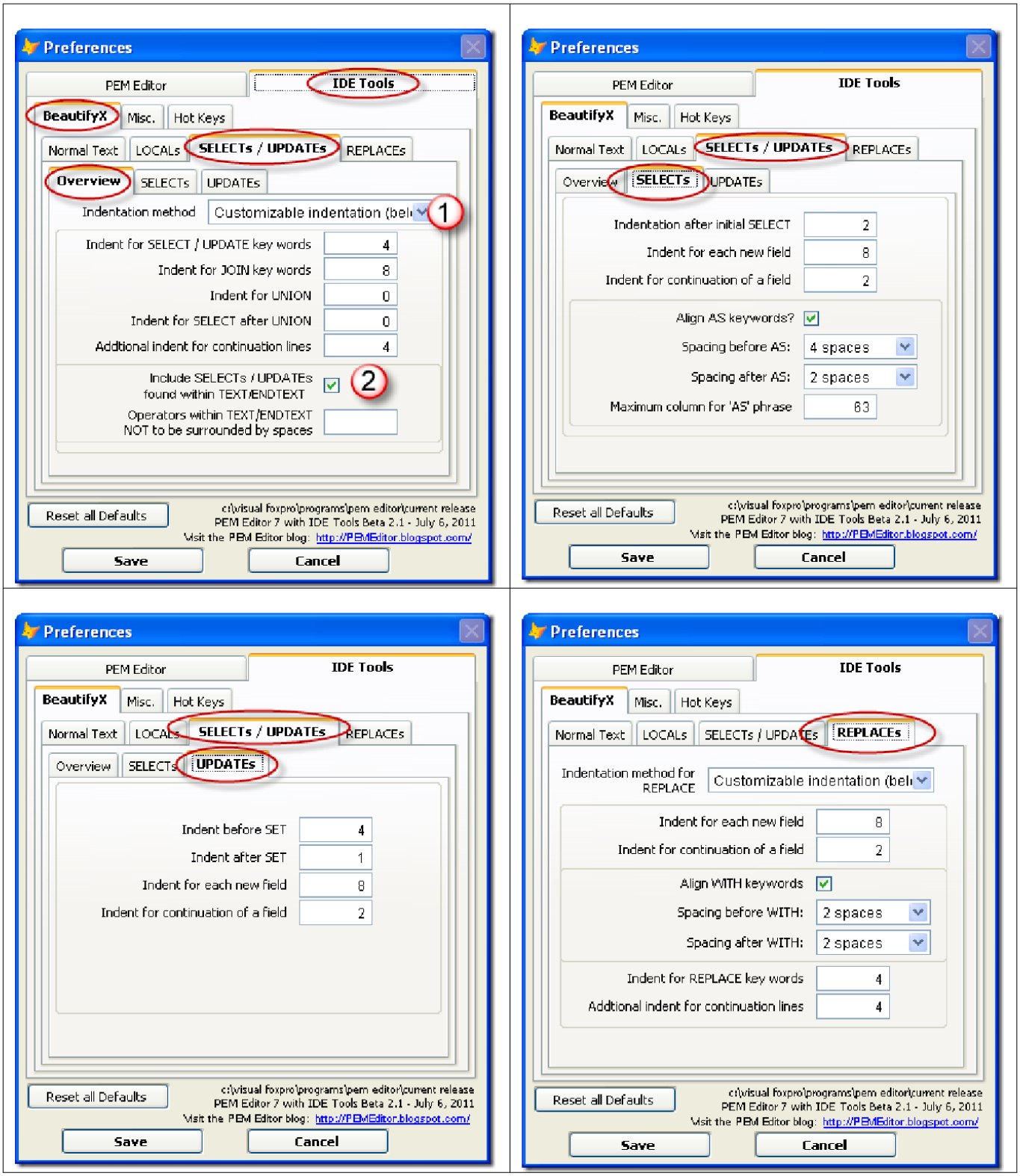
Two important things to take note of:

- For any of these options to take effect, you must select PEM Editor's customizable indentation.
- While Native FoxPro Beautify does not change lines in TEXT / ENDTEXT blocks, BeautifyX <u>can</u> apply this custom formatting of SELECT and UPDATE statements to such lines. There are two different mechanisms for doing so:
- There is a Preference item to apply SELECT, PDATE and DELETE to <u>all</u> TEXT / ENDTEXT blocks, as long as the first word is SELECT or UPDATE. This can be over-ridden for individual TEXT/ENDTEXT blocks by adding the directive {PEME:Ignore} (which is case insensitive) to the comments at the end of the TEXT line:

```
TEXT TO lcSQL TEXTMERGE NOSHOW    && {PEME:Ignore}
```

- Or, individual TEXT/ENDTEXT blocks can be formatted by adding the directive '{PEME:Select}' (case insensitive) to the comments at the end of the TEXT line:

```
TEXT TO lcSQL TEXTMERGE NOSHOW    && {PEME:Select}
```



Section 3.5.4. Extract to Method

This tool is used to extract highlighted text into a new method in the form or class being edited, as follows:

- Highlight the desired text.
- Run this tool.
- You will be prompted for the name of the method to be created. You may use mixed case names; _MemberData will be updated appropriately. This will cause the following:
- The new method will be created.
- The new method will be opened, populated with the highlighted text.
- In the original method, the highlighted text will be replaced with the appropriate reference to the new method, one of:
- Thisform.NewMethod()
- This.NewMethod()
- This.Parent.Parent … .NewMethod()
-

Section 3.5.5. Enhanced Cut / Copy

Applies to tools:

- Enhanced Cut
- Enhanced Copy
- Enhanced Cut / Additive
- Enhanced Copy / Additive

These are four closely related tools.

Enhanced Cut and *Enhanced Copy* work just like normal *Cut* (Ctrl+X) and *Copy* (Ctrl+C), except that when there is no selected text, they cut or copy the entire line the cursor is on.

The *Additive* variants of *Enhanced Cut* and *Enhanced Copy* will add the selected text (or, the entire line, if no text is selected) to the clipboard instead or simply inserting the text into the clipboard.

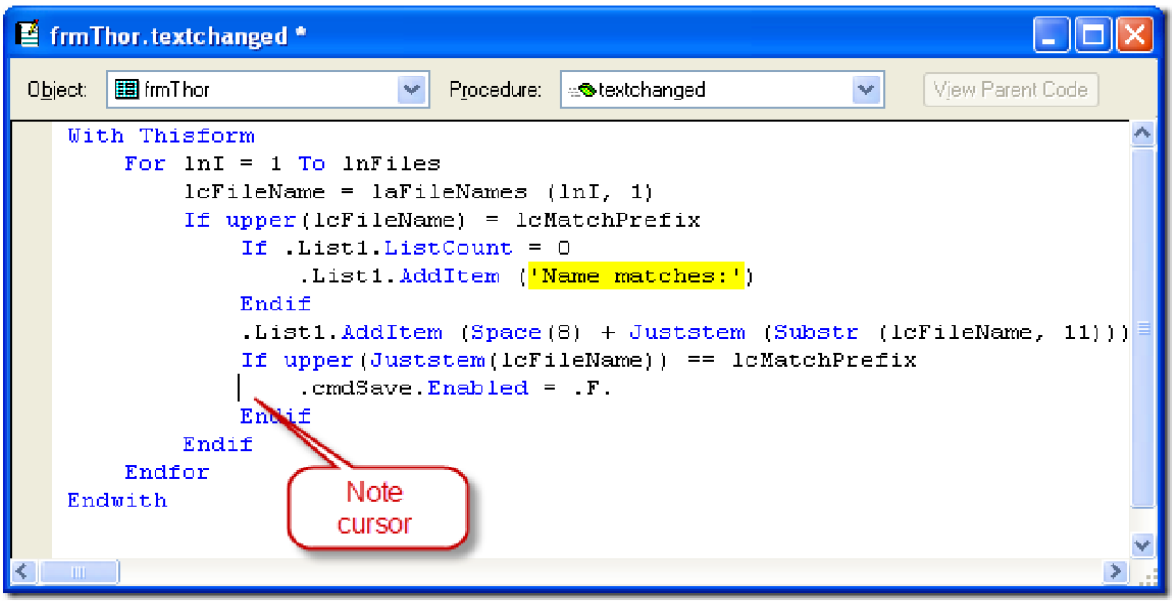
Section 3.5.6. Control Structures

3.5.6.1. Highlight Control Structure

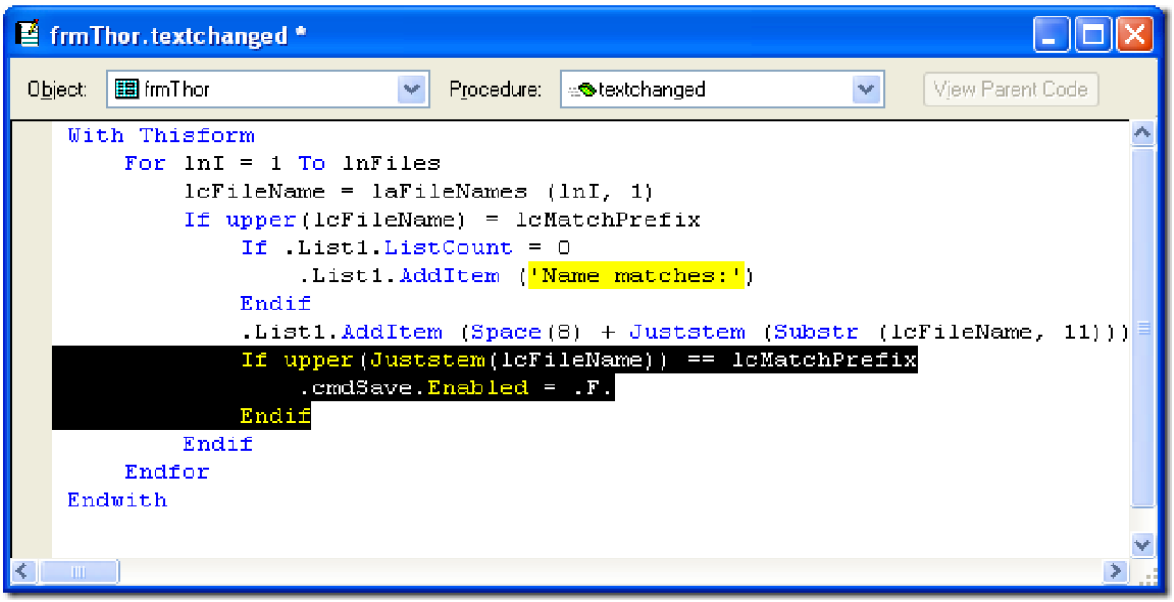
This tool highlights the current control structure where the cursor is. The cursor may be on the first or last line of the control structure, or any line within it.

This tool is closely related to another tool, "Close Control Structure".

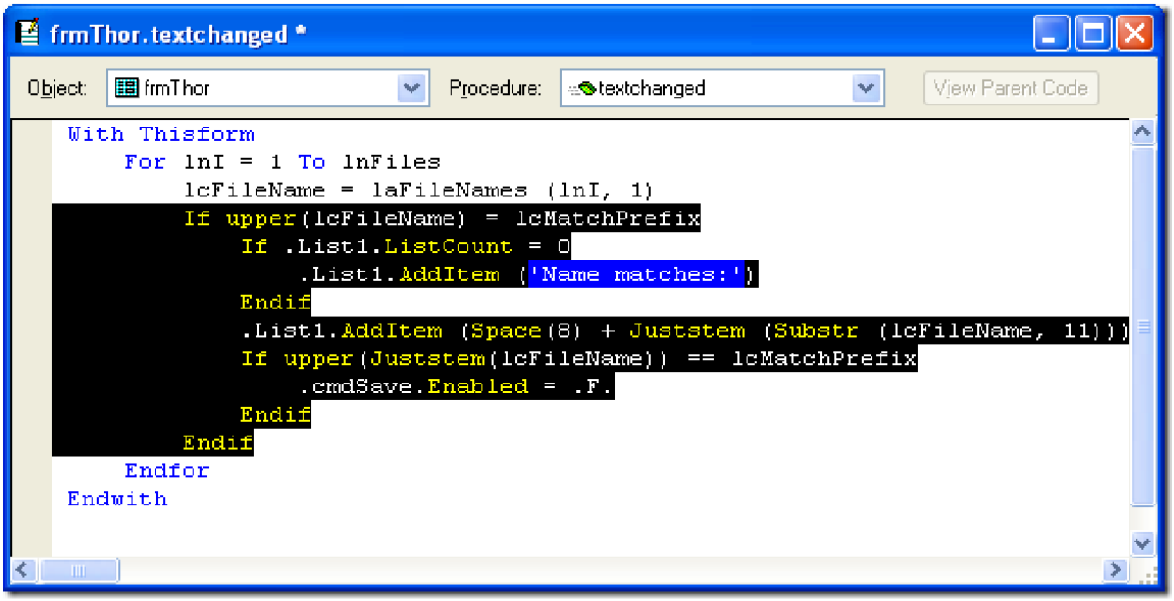
Thus, before:



And after:



Repeated uses result in ever wider control structures being highlighted:



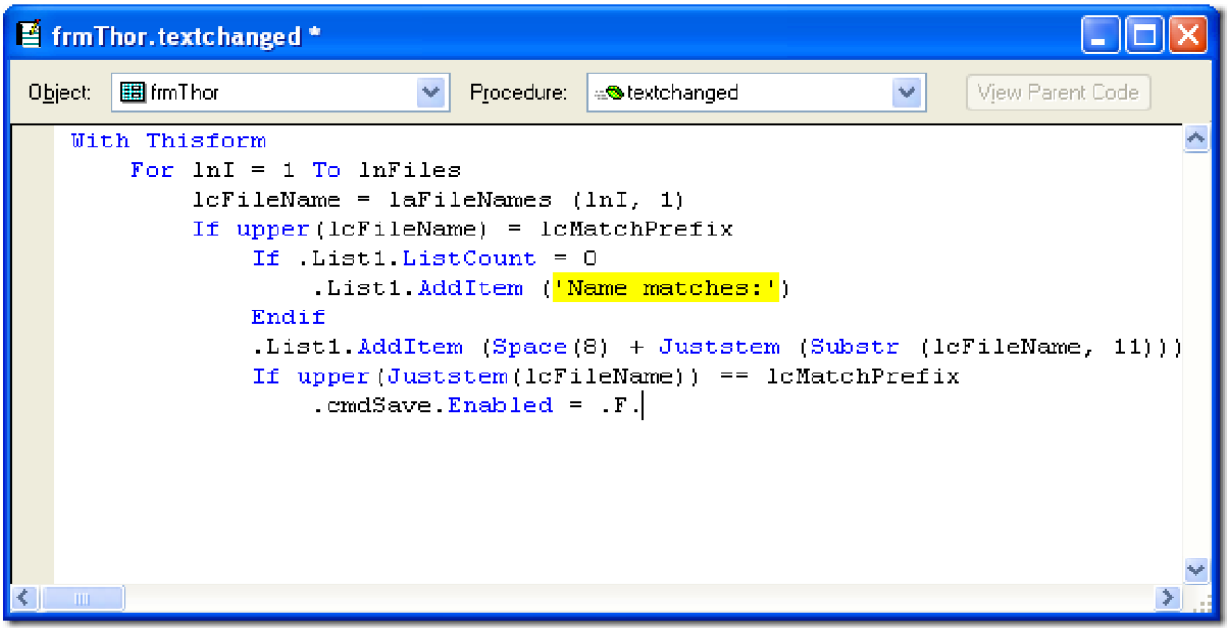
The Control Structures recognized are:

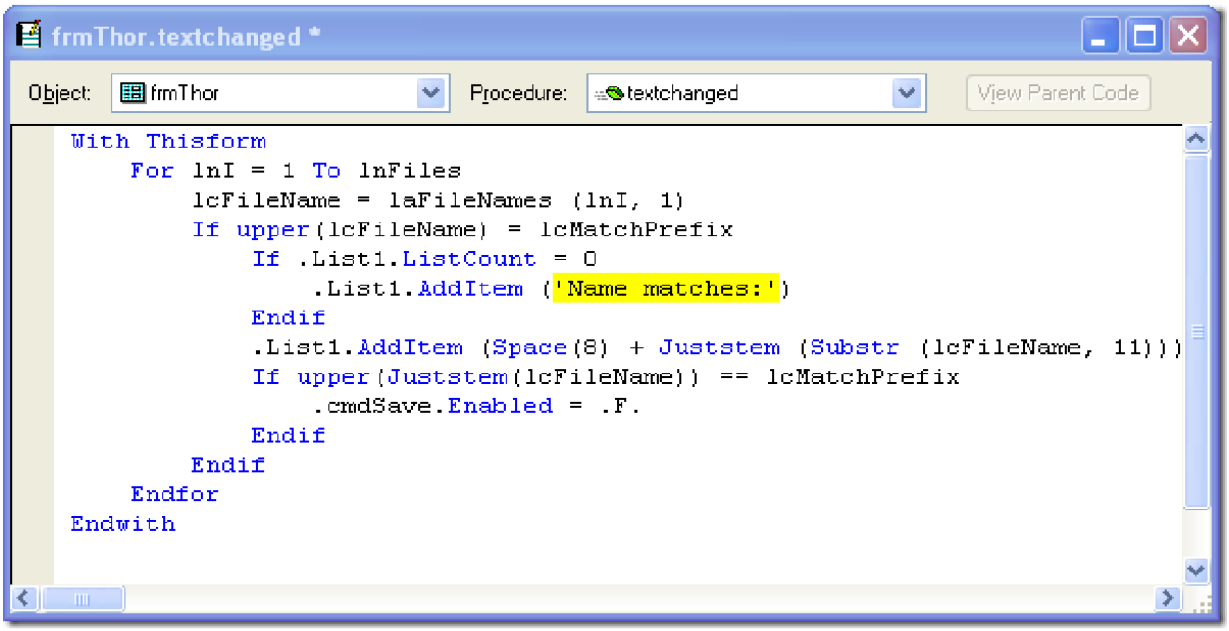
- If / Endif
- Do Case / EndCase
- Do While / EndWhile
- For / EndFor
- Try / EndTry
- With / EndWith
- Text / EndText
- Procedure / EndProc
- Function / EndFunc
- Define Class / EndDefine

3.5.6.2. Close Control Structure

This tool performs the simplest of tasks: it supplies the correct closing statement for a control structure.

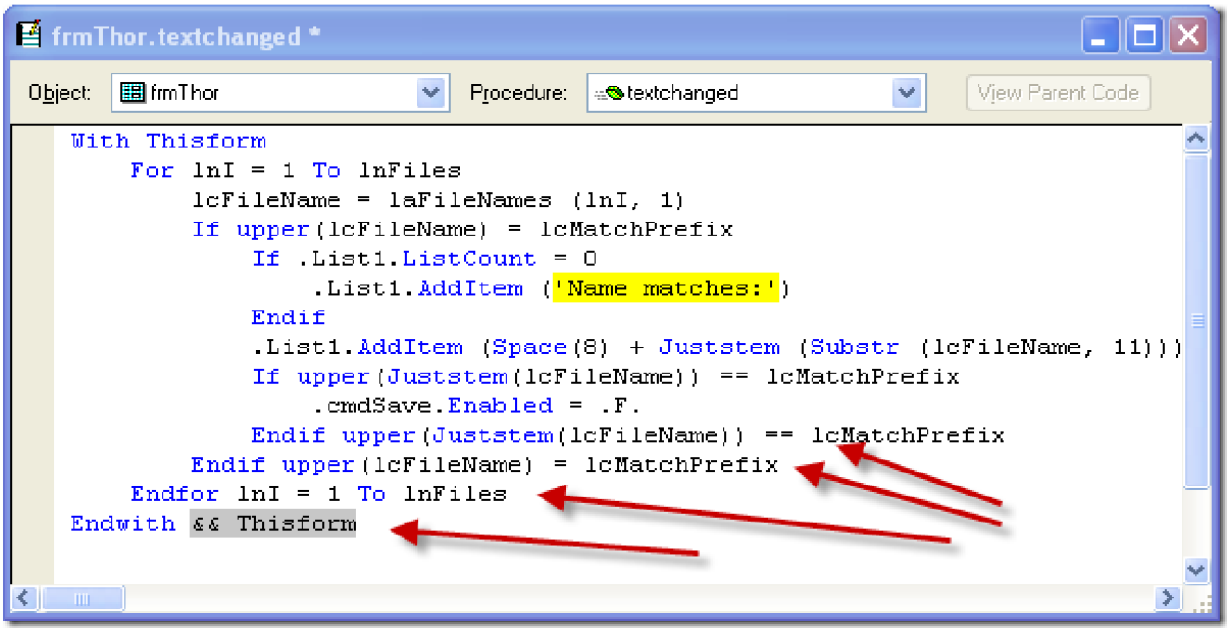
Follow the sequence as this tool is used repeatedly on this code sample, each time adding another closing statement.





Customization: There is a plug-In PRG that allows you to modify the behavior of this tool. It allows you to modify the text of the closing statement (such as including the text from a beginning IF statement). See Plug-Ins.

For example, see how this code could have looked if it used a Plug-In which includes the text from the beginning of each control structure:



Section 3.5.7. Code Listings

This applies to the following tools:

Code Listings:

- This control
    - All Code
    - All Code and Inherited Code
- This control and Children
    - All Code
    - All Code and Inherited Code

There's not much to explain about these tools; they all pop up a window with method code.

Some clarifications:

- 'This control' means:
    - The currently selected control being displayed in PEM Editor (if it is open)
    - or the currently selected control (as would be displayed in the Property Window). These two are almost always the same, but will differ at times
- 'Children' means: all the child objects of the control, down through the entire tree of objects.
- 'All Code' means all methods or events with non-default code
- 'Inherited Code' means more than you would expect. For any object which is a member of a parent class, this shows the code for the object in the parent class. Note that native VFP does not provide any mechanism for seeing this code. (See the sample below.)

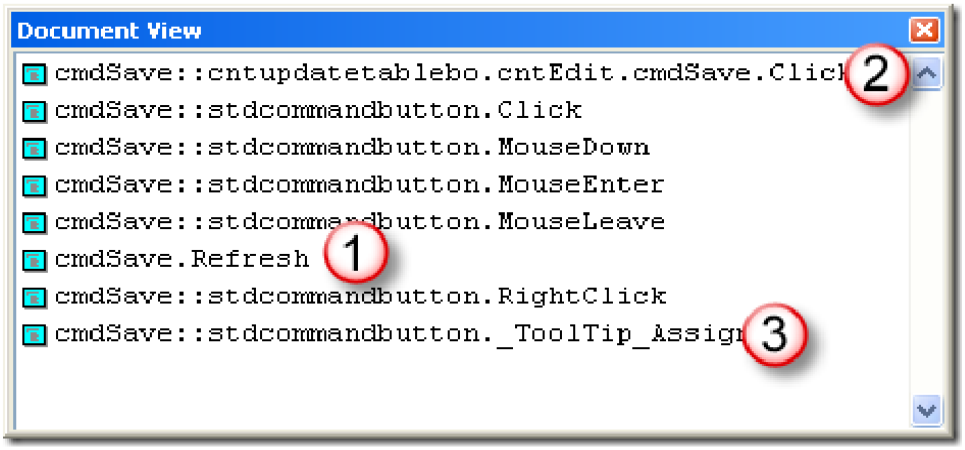
Note that the code is displayed in such a way that VFP Document View helps with navigation.

The example on the next page shows Document View for a code listing of a command button 'cmdSave' and its inherited code:

There is one event that has non-default code.

This object is a member of the parent class ‘cntupdatetablebo’, and you can see that there is a code for the click event there.

And there are a number of events that have code in the baseclass ‘stdcommandbutton’



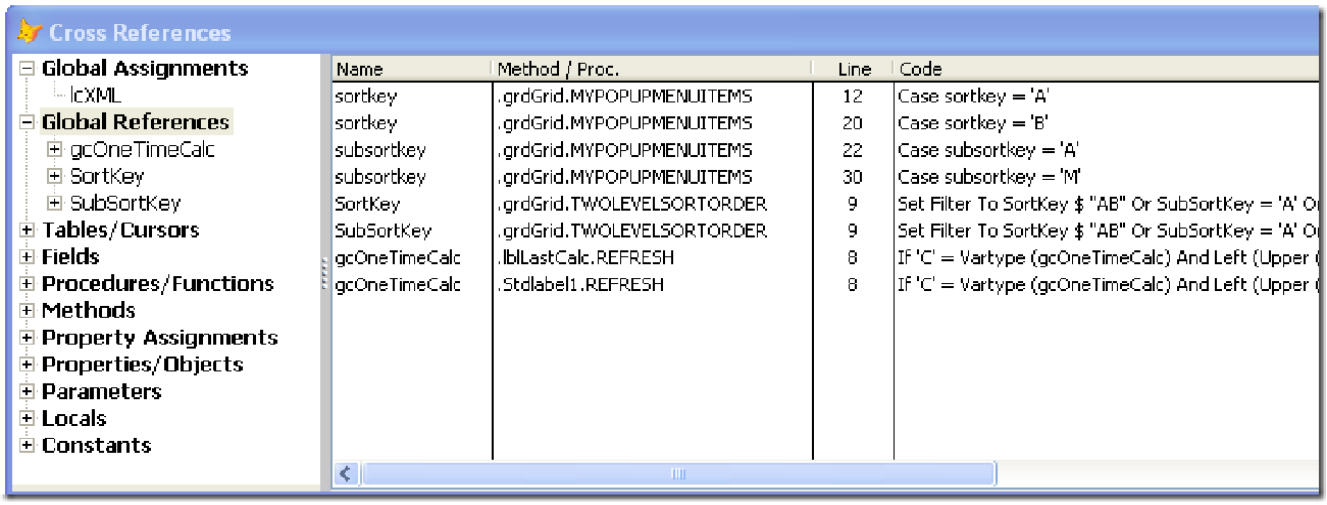
Section 3.5.8. Cross References

This applies to the following tools:

- *Cross References: This method*
- *Cross References: This form / class*
- *Cross References: This form / class and children*
- *Cross References: Current object*
- *Cross References: Current object and children*

Note that 'Children' here means: all the child objects of the control, down through the entire tree of objects.

The **Cross References** screen (see sample below) creates an analysis of all of the 'names' referenced in a PRG, method, or collection of methods. These names are broken down into about a dozen different categories; the TreeView on the left provides for navigation between the categories and the names within each category, and the grid on the right shows where the names are referenced. Double-clicking on a row in the grid opens the indicated method and highlights the relevant line of code.



Analysis / Categories in the TreeView: The Cross References screen breaks the 'names' referenced in code into about a dozen distinct categories. The 'names' actually include extended names, so that references to objects ('This.lblName.SomeProperty') or tables ('Customers.Name') are treated as single references. Furthermore, references within WITH/ENDWITH blocks are resolved to point to the object named in the WITH

statement. (This even applies to embedded WITH/ENDWITH blocks). The categories and their definitions are:

Global Assignments	Simple names that are assigned values, but are not locals or parameters
Global References	Simple names that are referenced, but are not locals or parameters
Tables/Cursors	References to names in VFP statements that indicate a table or cursor
Fields	References to names in SQL-Select statements
Forms	Do Form …
Procedures/Functions	Calls to PRGs, or procedures or functions within PRGs.
Methods	Calls to methods
Property Assignments	References to properties or objects that are assigned values
Properties/Objects	References to properties or objects that are not assigned values
Parameters	
Locals	
Constants	Compiler constants (#Define ...)

3.5.8.1. Warranty Information

This categorization cannot be perfect -- it is only at run time that VFP itself can properly identify which category any name falls into. While the first two and last three categories are completely reliable (global assignments and references, parameters, locals, and compiler constants), the remainder cannot be, and so there are instances where names will be reported in the incorrect category. For instance, 'Customers.Name' certainly looks like

a reference to a field in a table, but 'Customers' could (conceivably) actually be an object reference.

3.5.8.2. The Grid

The four columns may be moved or resized them as desired; their size and order will be remembered for next time. The grid displays one record for each line of code that a name is referenced on. If you double-click on one of the records, the method/PRG will be opened and the line of code will be displayed. There is a possible point of confusion when references are to code that is created on continuation lines. The analysis of code in this screen treats code on continuation lines as a single line of code. Thus, double-clicking to a reference in the grid will always highlight the first line in a set of continuation lines, even if the actual reference is to a later line in that subset.

3.5.8.3. Customization

There is a right-click context menu for the TreeView that allows for some customization:

You can select which of the categories are automatically expanded when the screen is first opened. (By default, all categories are collapsed).

You can select which single category is initially selected when the screen is first opened (causing the references to the items in that category to appear in the grid.

Section 3.5.9. Dynamic Snippets

Using Intellisense autocompletion, keywords entered into method code or in the command window can be expanded to create commonly used blocks of code: 'MC', for instance, becomes 'Modify Command', and 'DOCASE' expands to create a mutli-line DO CASE / ENDCASE block. This capability is familiar, very useful – but only creates static blocks of code.

Dynamic Snippets take this one step further: one or more keywords can be passed to a PRG to be processed so that the text that is then pasted in can be determined by the parameters.

Before the details, some examples:

CT Customers *… expands to …*

```
Use in (Select ("Customers"))
```

IT _oPEMEditor = O *… expands to ...*

```
If Type ('_oPEMEditor') = 'O'
```

UC lcName1 = lcName2 *… expands to …*

```
Upper(lcName1) = Upper(lcName2)
```

CASE lcType = 'Property', 'Event', 'Method' *… expands to …*

```
Do Case
   Case lcType = 'Property'
   Case lcType = 'Event'
   Case lcType = 'Method'
   Otherwise
EndCase
```

FP Alias *… opens a form listing all fields in {Alias}, allows selection of fields, and pastes in the selected field names.*

CT ('Close Table'), **IT** ('If Type'), **UC** ('Upper Case'), **CASE**, and **FP** ('Field Picker') are examples all Dynamic Snippet keywords.

3.5.9.1. How do Dynamic Snippets work?

Each Dynamic Snippet is defined by the existence of a PRG, having a particular structure for the name of the PRG, as well as for the parameters and result. Generally, the PRG will be called with the parameters that were entered into the code window, and the result is the text that is to be pasted in. *(More details below).*

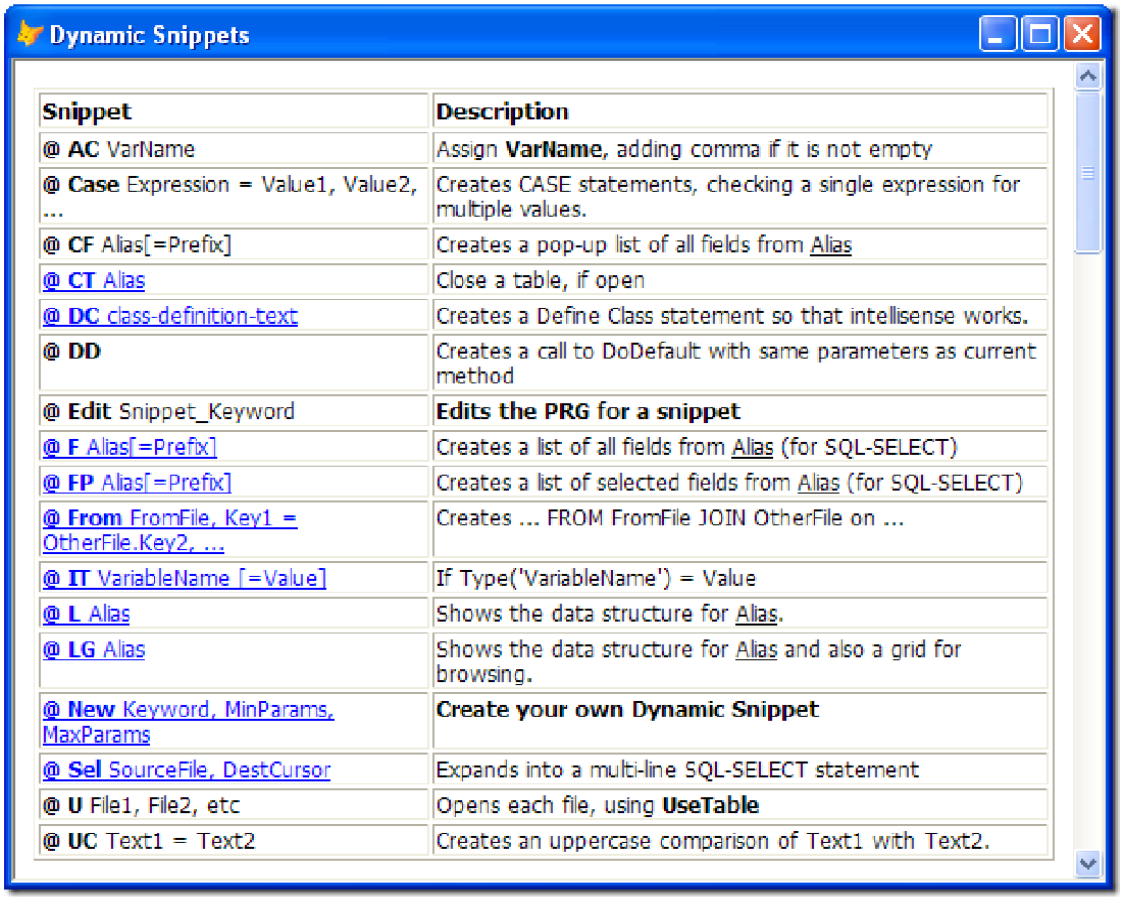
Dynamic Snippets are invoked either by using the assigned hot key (the recommended method) or selecting the Dynamic Snippet option from a menu (although this is a little cumbersome).

All text to the left of the cursor will be evaluated and passed to the appropriate PRG (with an exception, see below). The first 'word' must be the Dynamic Snippet keyword, followed by white space, and then the parameters which are to be passed, as text, to the PRG. *(More details below).*

Exception: As defined above, Dynamic Snippets could only be defined at the beginning of a line. They can, in fact, be entered elsewhere in a line. To indicate where the Dynamic Snippet keyword is to be found, use the Dynamic Keyword "identifier string", which has a default value of '@' (modifiable in the Preferences form). Thus, to use the **FP** Dynamic Snippet from the last example above, you would enter a line that looks like: **Select @FP Alias**.

3.5.9.2. What Dynamic Snippets are available?

There is an internal Dynamic Snippet '?', which brings up a form of all currently defined Dynamic Snippets. Thus, from the command window, enter '?' and using the Dynamic Snippet hot key will bring up a form that looks like this:



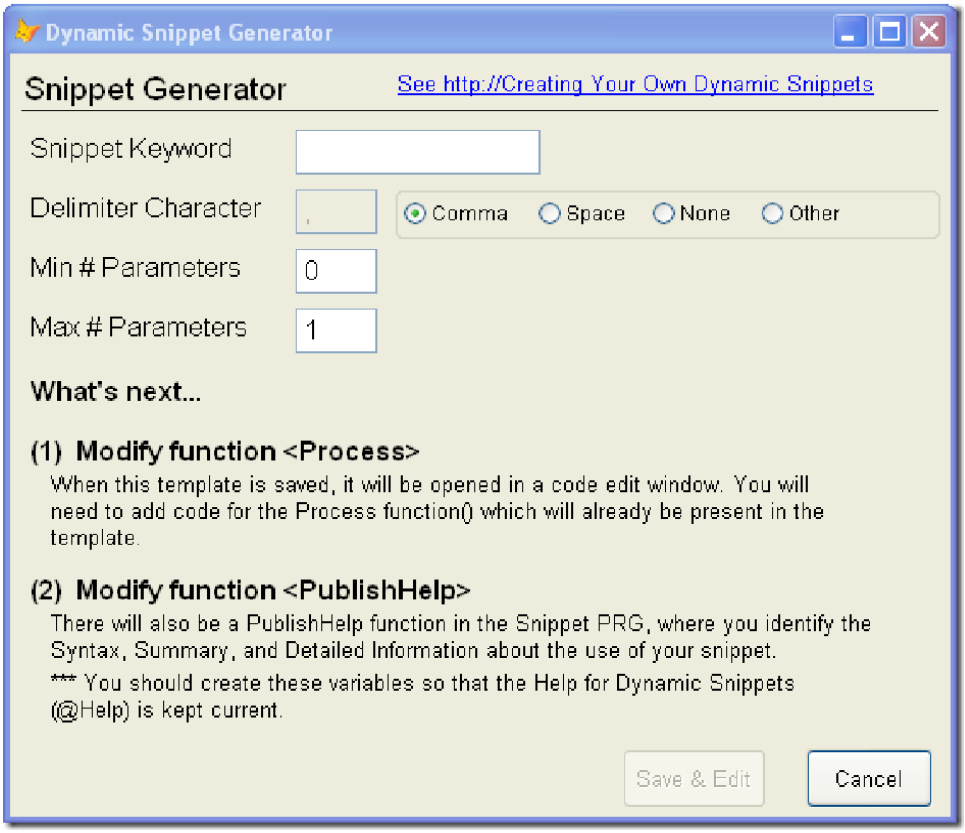
Snippet	Description
@ **AC** VarName	Assign **VarName**, adding comma if it is not empty
@ **Case** Expression = Value1, Value2, ...	Creates CASE statements, checking a single expression for multiple values.
@ **CF** Alias[=Prefix]	Creates a pop-up list of all fields from Alias
@ **CT** Alias	Close a table, if open
@ **DC** class-definition-text	Creates a Define Class statement so that intellisense works.
@ **DD**	Creates a call to DoDefault with same parameters as current method
@ **Edit** Snippet_Keyword	**Edits the PRG for a snippet**
@ **F** Alias[=Prefix]	Creates a list of all fields from Alias (for SQL-SELECT)
@ **FP** Alias[=Prefix]	Creates a list of selected fields from Alias (for SQL-SELECT)
@ **From** FromFile, Key1 = OtherFile.Key2, ...	Creates ... FROM FromFile JOIN OtherFile on ...
@ **IT** VariableName [=Value]	If Type('VariableName') = Value
@ **L** Alias	Shows the data structure for Alias.
@ **LG** Alias	Shows the data structure for Alias and also a grid for browsing.
@ **New** Keyword, MinParams, MaxParams	**Create your own Dynamic Snippet**
@ **Sel** SourceFile, DestCursor	Expands into a multi-line SQL-SELECT statement
@ **U** File1, File2, etc	Opens each file, using **UseTable**
@ **UC** Text1 = Text2	Creates an uppercase comparison of Text1 with Text2.

This list of all the Dynamic Snippets shows not only the keywords, but also their parameters and descriptions – and, if provided, a longer description as well (bookmarks at the left).

There are a number of pre-defined Dynamic Snippets. The actual code for them exists in the PEM Editor folder '**Dynamic Snippets\Snippet Samples**'. If desired, you can modify any of them and then activate your change by copying the PRG into the parent folder ('**Dynamic Snippets**').

3.5.9.3. How do I create a new Dynamic Snippet?

There is an internal Dynamic Snippet 'New', which brings up a form to initiate the creation of a new Dynamic Snippets. Thus, from the command window, entering 'New' and using the Dynamic Snippet hot key will bring up a form that looks like this:



This form allows you to indicate the Snippet Keyword (no spaces or special characters, since it will be used as part of the PRG name), what the delimiter character between parameters is to be, and finally provides instructions with how to proceed once you click "Save and Edit" (which creates the PRG and opens it for you to edit).

The PRG will be created in the '**Dynamic Snippets'** folder. You can move it, if you wish, to any folder in your path.

The PRG created will follow a very definite structure. You will put your code in the function <Process>. You will also want to modify function <PublishHelp> as well, since the information you enter there determines what is displayed when you review the list of all Dynamic Snippets (see above).

3.5.9.4. How can I edit Dynamic Snippets?

There is an internal Dynamic Snippet 'Edit' which takes a single parameter which is the snippet keyword you want to edit.

It is recommended that you actually use this on some of the pre-defined Dynamic Snippets to see how there are coded.

3.5.9.5. How are PRGs named and where do they live?

Dynamic Snippet PRGs are named <PEME_Snippet_Keyword.PRG>, where keyword matches the keyword for the snippet.

The PRGs can be stored anywhere in the path or in the '**Dynamic Snippets**' of PEM Editor. Note that if there are snippets found with the same name in multiple folders, the first one encountered takes precedence.

3.5.9.6. How can I share Dynamic Snippets?

Dynamic Snippets can be submitted to this address: VFPThorRepository@GMail.Com

Section 3.6. Customization: Plug-Ins

A number of plug-ins are available to alter the behavior of PEM Editor. Samples of each of them are provided in the sub-folder "Sample Plug-Ins."

Any plug-in can be enabled by either:

- Copying it into the sub-folder "Live Plug-Ins".
- Adding *PEME_* to the name of the plug-in (*PEME_GoToDefinition*, for instance) and saving it in your path.

Each sample PRG is amply commented, describing parameters and results.

AutoRenameControl	Determines names for objects when they are automatically renamed. All or part of the default behavior may be over-ridden.

CloseControlStructure	Determines the text to be used when closing a control structure (EndIF, EndWith, etc.) May be used to include the text from the opening of the control structure, if desired:
	`If This.lProperty`
	`  blablabla`
	`EndIf This.lProperty`

CreateLocalsStatements	Determines which variables assigned in a procedure are to be included in the LOCAL statements and also the format of the LOCAL statement. May be used to identify "orphan" locals. Called with an array of all possible locals.
GoToDefinition	Called after all other searches in GoToDefinition have failed, this allows further customization, such as trying to open the table for the name under the cursor.
OpenVCXFile	Called any time that any of the IDE Tools would open a VCX file. The default behavior is to open the class browser.

Section 3.7. MRU Lists and Source Code Control

This may seem like an unlikely pairing of features. We'll get to that in a bit.

Section 3.7.1. MRU Lists

MRU lists are the long-forgotten step-child of FoxPro. Effective tools when all work was done from the command window, they have fallen into disuse as reliance on the command window waned – the MRU lists are only accessible from the command window and are only updated for files opened from the command window.

This began to change with the introduction of PEM Editor. All forms and classes that are opened when PEM Editor is open are automatically added to their appropriate MRU Lists. PEM Editor even introduced a number of methods (now Thor tools) that either make the MRU lists available without using the command window (MRUs), or provide alternate ways to open files or classes which, as a side benefit, also automatically update the MRU lists (Open Files and Favorites).

Section 3.7.2. Source Code Control

There are two distinct issues which may be of interest to those who use Source Code Control (SCC)

- They may wish to be asked whether they want to check out files from their SCC Provider whenever the files are being opened.
- And some SCC Providers are case-sensitive with regards to file names, and FoxPro is notorious for being very flippant about changing the case of file names.

Section 3.7.3. To the rescue …

All files opened by PEM Editor and GoFish use the same method to open files, and this method performs three additional tasks:

- It adds the file to the appropriate MRU list (and, if the file is a class library, maintains a separate MRU list only for class libraries).
- It opens the file with the same case as it exists on disk, which implies that the case of the name will not be changed when it is saved.
- If the project uses Source Code Control and if the user desires, the user can be prompted as to whether to also check out the file from SCC.

This is not enough, though. Many people open their files using the project manager. These same capabilities listed above can also be made available for any file opened by the

Project Manager by placing the following code in the QueryModifyFile Event of the ProjectHook class for the project:

```
* Set the following to .T. if you want to be prompted
*     to check out files from SCC
#Define clCheckOutFromSourceCodeControl .F.

Lparameters oFile, cClassName
Local llSuccess, loTools

Try
      * see       http://vfpx.codeplex.com/wikipage?title=Thor%20Tools%20Object
      loTools      = Execscript (_Screen.cThorDispatcher, ;
      'class= tools from pemeditor')
      llSuccess = loTools.EditSourceX (oFile.Name, cClassName, , , , ;
clCheckOutFromSourceCodeControl)
Catch
      llSuccess = .F.
Endtry

If llSuccess
      Nodefault
endif
```

Tools from the Thor Repository

The Thor Repository is a catalog of tools collected from the members of the FoxPro community.

All FoxPro'ers are encouraged to submit IDE tools that they think would be of value to the FoxPro community at large (or, for that matter, to particular sub-sets of the FoxPro community). The tools do not need to be large or complicated but just useful (some already in the repository are only a few lines long.)

The Thor Framework (see the next section) provides easy access to a number of objects which make creation of certain types of tools a lot simpler. Tools using the Thor Framework are so noted in the table below.

Tools can be submitted to this address: VFPThorRepository@GMail.Com

As the intent of the Thor Repository is that it grow over time, the listing below of items in the catalog will be incomplete; follow the following link to obtain the most current catalog:

http://vfpx.codeplex.com/wikipage?title=Thor%20Repository

Tool	Description
Code	
Comment selected text	Improved way to comment selected text; also creates a comment line with date *(Framework)*
Insert class reference	Inserts a reference to the class that an object belongs to; This.Parent.Parent, etc. *(Framework)*
Insert color	Prompts for color, using GetColor(), and inserts RGB value *(Framework)*
Paste full name of object under mouse	Pastes the full path name of the object under the mouse into the code window. *(Framework)*
Go To ...	
Go To Method	Opens a dialog form to choose a method to view or edit. Allows searching by part of a method name. *(Framework)*
Go To Object	Select the object which the method belongs to. *(Framework)*

Miscellaneous	
Display ClassLibs	Display current class libraries in a dialog box, one per line.
Display Path	Display current path in a dialog box in a dialog box, one per line.
HackCX4 from MRU forms or classes	HackCX: Pop-up menu to select a form or class (from MRU lists) to be opened with HackCX4 *(Framework)*
Toggle comment colors	Toggle colors for comments in edit windows (normal vs. subdued)
Projects	
MRU classes in this project	Popup menu of MRU classes in the current project *(Framework)*
MRU files in this project	Popup menu of MRU lists for files in the current project *(Framework)*
Open project folder	Open an Explorer folder showing the folder of the active project
Samples	
Modify the "Last Revision" comment	Sample of finding a specific line in a edit window and modifying it. *(Framework)*
Wrap text with IF / ENDIF	Sample of taking the selected text in an edit window, modifying it, and pasting back the replacement. *(Framework)*
Tables	
Create SQL from cursor	Puts a CREATE CURSOR SQL statement on the clipboard from an open cursor.
Schema	Print/View the current table's schema
WLC Column Listing Utility	LIST FIELDS replacement by White Light Computing.
Windows	
Cycle thru Designer windows	Cycle thru Form Designer and Class Designer windows. *(Framework)*
Move Designer	Move Form and Class Designers to top of screen and align them

windows to top	horizontally. *(Framework)*
Resize Designer Window	Resize current Form / Class Designer window to show the entire form or class being edited. *(Framework)*

Section 4. The Thor Framework

Thor provides a framework of tools to assist in creating tools.

Each of the tools in the framework can be obtained from a single line of code that looks like this:

```
Result = Execscript (_Screen.cThorDispatcher, cParameter)
```

This rather unusual approach was chosen because it achieves two goals:

- The tools in the framework are available with only the single dependency (the name of the property in _Screen) and thus can be accessed by tools regardless of the folder where Thor is installed.
- The tools in the framework are available even after a 'Clear All'.

Section 4.1. Internal Tools

cParameter	**Result**
Run	Runs Thor. Same as Do Thor with 'Run' or Do RunThor
Edit	Opens Thor form. Same as Do Thor with 'Edit'
Clear HotKeys	Removes all Thor-assigned keyboard macros so that a macros (FKY) file can be saved
Tool Folder=	Returns the name of Thor's tool folder
Thor Engine=	Returns an object (documentation not yet available)
Thor Register=	Returns an object so that an APP can self-register its own tools, such as is done by GoFish4 and PEM Editor 7.
Class= ContextMenu	Returns an object used to create context menus. (documentation not yet available)
Class= ThorFormSettings	Returns an object so that forms can save their settings (size, position, etc). (documentation not yet available)
Class= FindEXE	(documentation not yet available)

Section 4.2. External Tools:

The structure of Thor provides for the inclusion of other objects embedded in APP files. In particular are these two, developed as part of PEM Editor.

cParameter	Result
Class= editorwin from pemeditor	Methods to access and modify the text in the currently open editing window (Select, Cut, Copy, Paste, etc ...) - see the section below, 'Thor EditorWindow Object'
Class= tools from pemeditor	A collection of various methods, not related to each other, but of value beyond their use in PEM Editor – see the section below, 'Thor Tools Object'

Section 4.2.1. Thor EditorWindow Object

All of the IDE features of PEM Editor which access and / or modify the text in the currently open editing window use the functions from **FoxTools.fll**.

The Thor EditorWindow Object is made available to make it simple to build Thor tools which can also access or modify the text in the currently open editing window. This object serves two purposes:

- It provides wrapper methods for the most useful functions in **FoxTools.fll**.
- When accessed, it has already determined the handle for the current editing window. All of its methods reference this handle, eliminating the need for the handle to be otherwise known or referenced.

The Thor EditorWindow Object can be obtained from this single line of code:

```
loEditorWin = Execscript (_Screen.cThorDispatcher,'class= editorwin from pemeditor')
```

Some further notes:

- Character positions and line counts start at 0, not 1. (i.e., be careful)
- While this object only is available if PEM Editor has been installed, the PEM Editor form itself need not be open for it to work.

There are numerous uses of this object in the tools in the Thor Repository.

4.2.1.1. Window manipulation: size, position, title, etc.

Methods (Parameters)	Description
CloseWindow()	Close the current window
FindLastWindow()	Returns the handle of the most recently used window which is either of a PRG or method code from the Form or Class Designer.
FindWindow()	Saves the handle for the currently active window, and returns its window type: 0 Command Window, Form and Class Designers, other FoxPro windows 1 Program file (MODIFY COMMAND) 2 Text Editor (MODIFY FILE) 8 Menu code edit window 10 Method code edit window of the Class or Form Designer 12 Stored procedure in a DBC (MODIFY PROCEDURE) -1 None
GetHeight()	Returns the height of the editing window, in pixels.
GetLeft()	Returns the left position of the editing window, in pixels.
GetOpenWindows()	Returns a collection of the handles of all open windows, most recently used first.
GetTitle()	Returns the title for the current window
GetTop()	Returns the top position of the editing window, in pixels.

GetWidth()	Returns the width of the editing window, in pixels.
GetWindowHandle()	Returns the handle of the current editing window
MoveWindow (tnLeft, tnTop)	Moves the editing window to position {tnLeft}, {tnTop}. Both are in pixels.
ResizeWindow (tnWidth, tnHeight)	Resizes the editing window to {tnWidth} by {tHeight}. Both are in pixels.
SelectWindow (tnHandle)	Selects (brings to the foreground) window with handle {tnhandle}
SetHandle (tnHandle)	Sets the handle (used to indicate the window being referenced in most of these commands/)
SetTitle (tcNewTitle)	Sets the title for the editing window to {tcNewTitle}

4.2.1.2. Text manipulation

Methods (Parameters)	Description
Copy()	Copies the currently highlighted text into the clipboard
Cut()	Cuts the currently highlighted text
EnsureVisible (tnPosition, tlScroll)	Ensures that the character at position {tnPosition} is visible in the editing window. If {tlScroll} is true, it is brought to the mid-point of the editing window
GetCharacter (tnPosition)	Returns the character at position {tnPosition}
GetEnvironment {tnIndex}	Returns a single environment setting. {tnIndex} takes values from 1 to 25. See _EdGetEnv in the help for FoxTools for a description of all the settings. Frequently used values are: 2 File Size

	17 Selection start 18 Select end 25 Window Type
GetFileSize()	Returns the file size
GetLineNumber (tnSelStart)	Returns the line number for the character at position {tnPosition}
GetLineStart (tnSelStart, tnLineOffset)	Determines the line number for the character at position {tnPosition} and returns the position for the character at the beginning of that line. If {tnLineOffset} is supplied, it first offsets the line number by that amount. Thus .GetLineStart (tnSelStart, 1) gives the start position of the next line after the line for {tnSelStart}
GetSelEnd()	Returns the position for the end of the currently highlighted text
GetSelStart()	Returns the position for the start of the currently highlighted text
GetString (tnSelStart, tnSelEnd)	Returns the string of characters from position {tnSelStart} through {tnSelEnd}
Paste(tcText)	If {tcText} is supplied, pastes it into the editing window, leaving _ClipText unchanged. Otherwise, pastes the contents of the clipboard into the editing window.
Select (tnSelStart, tnSelEnd)	Selects (highlights) the string of characters from position {tnSelStart} through {tnSelEnd}
SetInsertionPoint (tnPosition)	Sets the insertion point to {tnPosition}

Section 4.2.2. Thor Tools Object

This object is a collection of various tools, created for PEM Editor, which have applicability as well outside of the PEM Editor form.

The Thor Tools Object can be obtained from this single line of code:

```
loTools = Execscript (_Screen.cThorDispatcher, 'class= tools from pemeditor')
```

Methods (Parameters)	Description
AddMRUFile (tcFileName, tcClassName)	Adds a file to its appropriate MRU list (in the FoxPro resource file). If the file is a class library, but no class name is supplied, adds the file to the MRU list for class libraries (unique to PEM Editor)
CloseForms()	Close the PEM Editor and Document TreeView forms, if open
CreateNewPEM (tcType, tcName, txValue)	Creates a new property or method: tcType = 'P' for Property, 'M' for Method tcName = Name for new PEM (_MemberData if appropriate txValue = Value (for properties) or method code (for methods)
DiskFileName (tcFileName)	Returns the file name as it is stored on disk (that is, with current upper/lower case).
EditMethod (toObject, tcMethodName)	Opens a method (or event) for editing. {toObject} may be an object reference, .T. for the current form or class empty for the current object.
EditSourceX (tcFileName, tcClass, tnStartRange, tnEndRange)	Opens any file for editing (as **EditSource** does), with additional capabilities: The file is added to its appropriate MRU library. The file is opened with the correct case for the file name, so that when it is saved the case for the file name will not

	be changed. If the file is a class library, and no class name is supplied, the class browser is opened. You will be asked whether you want to check out the file from Source Code Control (if you use SCC and you have marked the appropriate item in the Preferences file.)
FindObjects (tcSearchText)	Finds all objects matching the search criteria in {tcSearchText}. The search criteria are the same as are specified by the 'Find' (binoculars) search button. The result is a collection, where each item in the collection is an object with two properties: Object - a reference to the object FullObjectName - the full path name to the object *For example,* ***.FindObjects ('Exists ("ControlSource")')*** *returns a collection of all objects having a ControlSource.*
FindProcedure (tcName)	Finds a PRG named {tcName}, or a procedure or function named {tcName} within a PRG, or a constant named {tcName}, opens the file for editing, and highlights the searched-for name.
GetCurrentObject (tlTopOfForm)	If (tlTopOfForm) is true, returns the current form/class. Otherwise, returns the currently selected object.
GetFullObjectName (toObject)	Returns the full name path of an object {toObject}
GetPEMList (toObject, tcTypes)	Returns a collection of the names of PEMs for an object. {toObject} may be an object reference .T. for the current form or class empty for the current object.

	{tcTypes} may be one or more of 'P' (for properties), 'E' (for Events), 'M' (for Methods) or, if empty or missing, the collection will contain all PEMs.
GetMRUList (tcName)	Returns a collection of file names in a MRU list. {tcName} may be a file name, a file extension, or the actual MRU-ID (if you know it)
GetThis()	Returns the object that the current method belongs to.
SelectObject (toObject)	Selects {toObject} as the current object displayed in the PEM Editor form and in the Properties Window (if possible).
UseHighlightedTable()	SELECTs or USEs the current highlighted file. (If there is no highlighted text, uses the alias that the cursor is in). If the alias exists, it SELECTs it. Otherwise, it looks for {PEME_OpenTable.PRG} and executes it, assuming that it will open the table (if possible) An example of {PEME_OpenTable.PRG} can be found in the sub-folder **Dynamic Snippets\Snippet Samples** of the installation folder for PEM Editor. You can activate this by copying it to its parent folder (make any adjustments to it as you need.)