

Get More Productive with Visual FoxPro

Session Number rsch2

*Richard A. Schummer
President*

*White Light Computing, Inc.
42759 Flis Dr.*

Sterling Heights, MI 48314

Voice: 586.254.2530

Fax: 586.254.2539

E-mails: raschummer@whitelightcomputing.com

rick@rickschummer.com

Web sites: www.whitelightcomputing.com

www.rickschummer.com

Overview

Rapid Application Development is a leftover buzzword from the 90's. Are you as productive with VFP as you can be, or wish to be? How do other developers use the world's best database application development tool to bring applications to market quicker? Are there tips I can learn to save me 10 minutes a day or an hour a week? This session will demonstrate as many tips and productivity ideas that can be crammed into a 75 minute session.

As the old saying goes, there are always three ways to accomplish something in Visual FoxPro. Sometimes we only know one way and there are two other ways that are faster or better. Sometimes we don't even know that you can accomplish certain things with the VFP. I am constantly amazed, even after using Visual FoxPro for more than seven years, how much I have learned just looking over the shoulder of others as they develop with this product.

VFP 8 (Toledo), while focused on features that effect the end user's experience, has a number of excellent productivity enhancements. The big ticket item in VFP 9 (Europa) is the Report Designer, but again the Fox Team has added some excellent goodies that help developer productivity. The session will have productivity tips for developers working with all versions of VFP, including the marketing beta of VFP 9 (which attendees take home from this conference).

Session attendees can learn...

1. How to make better use of the VFP tools: Class Browser/Component Gallery, IntelliSense Manager, Object Browser, and Task List Manager, plus VFP 8/9's Task Pane, Toolbox, and Code Reference.

2. How to make better use of the code editors and IntelliSense.
3. Why features like Document View, List Members, Quick Info, and Macros should be at your fingertips.
4. How changes to the Property Sheet will help you be more productive.
5. How to extend the Class and Form Designers with Builders.
6. How to use the Debugger more effectively (including VFP 8/9 enhancements).
7. How to enhance the VFP IDE to be more productive day-to-day.
8. How to make better use of the VFP Debugger.
9. How to use developer tools available that can also increase your productivity.

Skill Level/Prerequisites

Introductory to Intermediate developers will learn the most, but all developers can pick up tricks to add to their day-to-day development. There are no prerequisites to this session.

Table of Contents

Overview	1
Session attendees can learn.....	1
Skill Level/Prerequisites.....	2
Table of Contents	2
VFP Tools.....	6
Class Browser/Component Gallery	6
Set the default file to be opened when Class Browser is started (<i>ClassBrowserTips.prg</i>)	6
Opening the Class Browser with a specific class (<i>ClassBrowserTips.prg</i>)	6
Moving and copy classes between class libraries	7
Renaming methods and properties without opening the class	7
Safely change a class name without breaking references to subclasses	7
Test classes from the Class Browser	8
Viewing and editing superclass code via the Class Browser	9
Class Browser can open/maintain PRG based classes (VFP 9) (<i>CBPRGClasses.prg</i>)	9
Class Browser Add-in to retain the Regional Settings for time and date (<i>CbChangeDateFormat.prg</i>).....	10
Project Manager.....	11
Drag-n-Drop	11
Dragging from one project to another	11
Drag objects from a project to a designer	11
Dragging from project to program code	12
What happens when dragging from Class Browser or Component Gallery to a project?	12
Dragging from a non-VFP application to a project?	12
Open classes in Class Browser (VFP 9)	13
Build messages recorded immediately and to Debug Output window (VFP 9)	13
Change Font in Project Manager (VFP9)	13
Task List	14

Adding my own custom fields to the Task List? (<i>TLExt.dbf/fpt</i>)	14
Using my custom fields in the Visual FoxPro Task List	16
Fixing the Task List when it seems to have lost its mind	16
Task List tasks disappear after I add an existing user-defined fields table.....	17
Putting it all together with the WLC Task List Editor (<i>Example: TLEditor.scx/sct</i>).....	17
Object Browser.....	18
Determining the values of constants defined in a COM object	18
Use the Object Browser to create class templates to implement interfaces	19
Find out the name of the OCX file to ship with my deployment setup	19
Task Pane	19
Add shortcuts to your developer tools.....	20
Shortcuts to key VFP news sites.....	20
Managing the VFP environment.....	21
Field Mapping in Environment Manager (VFP 9)	22
SQL Data Explorer (VFP 9)	22
Exploring Web services.....	23
Fully Configurable.....	24
Toolbox	24
Text Blocks.....	25
ActiveX Controls.....	25
Dockable (VFP 9).....	25
Code Reference	26
Editors	27
Document View.....	27
Beautify Code Blocks (VFP 8).....	27
Printing Source Code Files and Syntax in Color (VFP 8).....	27
Pasting code retains colors (VFP 9)	27
Background compilation (VFP 9).....	28
Select printed text (VFP 9)	29
SET PATH ... ADDITIVE (VFP 9) (<i>Example: SetPathAdditive.prg</i>)	29
Bookmarks.....	29
Creating/Removing Bookmarks (toggle).....	29
Creating/Removing Shortcuts (toggle)	30
Moving between shortcuts.....	30
Gotchas	30
Property Sheet	30

Hotkey to locate property/method	30
Zoom window.....	30
Member Data (VFP 9)	30
Favorites (VFP 9)	32
Default values for New Property (VFP 9).....	33
Font selection (VFP 9).....	33
IntelliSense	33
ActiveX Controls.....	33
Custom expansion (VFP 8) (<i>Example: IntellisenseCustomExpansion.prg</i>)	34
List Members, Quick Info	34
C++ Operators	35
Works inside of WITH...ENDWITH & FOR EACH (VFP 9).....	36
_VFP.EditorOptions property persistent via FoxUser (VFP 9).....	36
Designer Tips	37
Menu Designer	37
Support for moving menus (VFP 8)	37
Support to hook in your own Menu Designer (VFP 9)	37
Builders	37
Referential Integrity (RI)	38
Option Group.....	38
DataEnvironment (VFP 8).....	38
CursorAdapters (VFP 8).....	39
Create your own builder	39
Write a program (<i>GenProjecthook.prg</i>)	39
BuilderX (<i>BuilderDemo2.scx, demo.vcx::frmCheckboxBuilder</i>)	40
BuilderB/BuilderD.....	41
Tab Order (VFP 8)	42
Tab Order on properties for form/class designer (VFP 9).....	42
VFP Design Surface (VFP 8)	43
No maximum limit for design area.....	43
Debugger	43
Set the debugger configuration to factory settings	43
Save and restore the configuration of the debugger	45
Change values of memory variables in the debugger.....	46
Drag and Drop	46
Changing Expressions	46

Quick access to the property values of a specific object	46
IntelliSense in the Watch window (VFP 8)	47
Coverage Profiler.....	47
VFP IDE	48
BROWSE	48
BROWSE memo tips (VFP 9).....	48
Macros / ON KEY LABEL	49
Dockable forms for custom tools (VFP 9).....	50
SHIFT key and menus	50
C5 Errors slowing you down?	51
Use a developer menu to run repetitive tasks	51
Mopping up after a program crash	52
Developer Tools	52
G2 SuperCls.....	52
RAS ProjectBuilder	53
WLC Tab Order Builder.....	54
Conclusion.....	55
Special Thanks.....	56
Copyright.....	56
Author Profile.....	56

VFP Tools

Each release of Visual FoxPro brings more and more of the tools that Microsoft refers to as the “Xbase tools”. These tools include the Class Browser/Component Gallery, IntelliSense Manager, Object Browser, and Task List Manager, plus VFP 8’s Task Pane, Toolbox, and Code Reference. These tools all have important features that make us productive and provide us with functionality not found in the various editors and designers. This white paper will address a number of tips for each of these tools.

Class Browser/Component Gallery

The Class Browser is a powerful tool that helps a Visual FoxPro developer manage classes and class libraries. From the Class Browser you can create, modify, delete, and subclass a class. You can redefine the class superclass, and change the icon that is displayed in the Project Manager and Class Browser. Gaining a full grasp of the capabilities of the Class Browser can reap important benefits for a Visual FoxPro developer.

The Component Gallery tool displays shortcut icons to components available to your Visual FoxPro projects and designers. This enables you to manage, in various ways, any file available to your system. It is one way to catalog and organize classes (not necessarily from the same class library) together. The Component Gallery also integrates with the VFP Application Builder for those developers brave enough to work with the VFP Application framework.

Set the default file to be opened when Class Browser is started

(ClassBrowserTips.prg)

Ever have one of those days where you work on refactoring a class or developing a set of classes in the same class library all day? You keep opening up the Class Browser and picking the same class library over and over. Ever wish that you could set it up so that when you open up the Class Browser it opens up a specific class library? Well you can if you set it up to do so.

First open up the Class Browser and open up the class library that you want to be the default class library opened when it starts. In the Command Window:

```
_oBrowser.SetDefaultFile()
```

If you want to clear the default class library execute the following statement:

```
_oBrowser.ResetDefaultFile()
```

Opening the Class Browser with a specific class

(ClassBrowserTips.prg)

When starting up the Class Browser programmatically, you can open to a specific class and even preselect the class and a property or method of the class.

The first parameter passed to the Class Browser is the class library. The class library must be on the Visual FoxPro path or be full qualified to have the class library open successfully. The second parameter is the class name, followed by a period and then the property or method. If you just pass the class name as the second parameter the class library will be open, but no class is selected. You need to pass a property or method attached to the class name to get the class highlighted in the left pane (TreeView). Here is an example call to the Class Browser to open up with the *lCopyAppToTestDirectory* property and the *phkDevelopment* class selected in the *CPhkBase2* class library.

```
DO (_browser) WITH "CPhkBase2", " phkDevelopment.lCopyAppToTestDirectory"
```

This can be handy if you are trying to save time opening up the same class library over and over during a development session. Another shortcut is to start the Class Browser, right-click on the open commandbutton and select a class library from the most recently used list that drops down.

Moving and copy classes between class libraries

Using two instances of the Class Browser allows developers to move and copy classes between class libraries. The key to moving or copying classes is to drag the class from the class icon in the top left corner of the Class Browser. It is important to hold the Ctrl key down first before selecting the icon. Drag-and-drop between class browser instances defaults to moving the class. To copy the class, hold the Ctrl key down during the drag to the second instance. You can drop the class on any part of the second Class Browser.

Renaming methods and properties without opening the class

You can rename methods and properties by right-clicking on the property or method and selecting Rename... on the shortcut menu. A dialog is presented which allows you to rename the property or method. Please remember that renaming a method or property does not magically rename the references in the method code. You will need to manually search and replace any references that are changed (there is one caveat discussed later when manually replacing references is unnecessary).

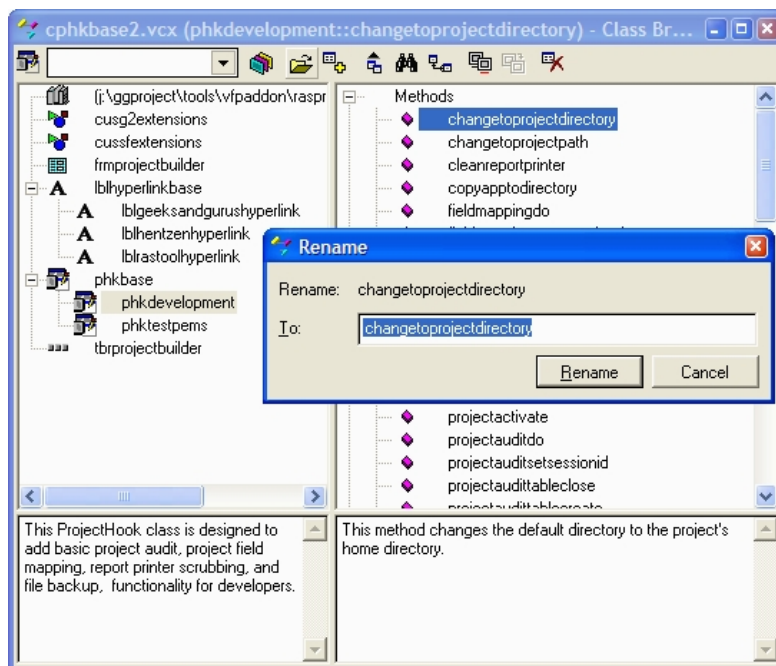


Figure 1. The Class Browser provides a method and property renaming feature without opening up the class.

Safely change a class name without breaking references to subclasses

It never seems to fail. You start out developing this cool class and you have it in a class library that at 3:00 in the morning seemed to make sense. You develop it, you tweak it, and after significant testing you implement the class in a number of forms and other classes in a project. The next morning, after getting some needed sleep, and after that first can of Coke you realize that a different class library is the more appropriate location for the class. Moving or renaming a class or a class library can cause all kinds of trouble for Visual FoxPro developers. If you rename a class, or move a class, all the other classes and forms that contain this class will ask you to locate the class each time you open them in the designers. This can be a big pain, especially if the classes being renamed are an integral part of a framework. So how can we avoid such pain?

The Visual FoxPro Class Browser will adjust the name and/or location of all class library and form files in the Class Browser automatically if a subclass or instance exists for the class changed. You can also open Visual FoxPro project files, which loads in all class library and form files for the project. Therefore, if you rename a class it is

recommended that all class library and form files that use or are a subclass of the changed class are loaded in a Class Browser window to have the reference automatically updated.

One thing that it will **not** automatically change is code that references the class like:

```
thisform.oRegistry = NEWOBJECT("cusRegistry", "CFramework")
this.oBusiness      = CREATEOBJECT("cusInvoiceBO")
```

The Class Browser is only smart enough to fix the object inheritance hierarchy, not references to the classes in code. This is something you will need to handle manually. But it is much better than handling the code and the objects that are broken because the name of the class was changed. If the class is from a common library and used across several projects, you want to be sure to open up each of the projects affected, or reconsider your decision to rename the class.

Test classes from the Class Browser

The Visual FoxPro Class Browser has some terrific drag and drop capabilities that assist Visual FoxPro developers in testing classes. The various scenarios will allow developers to instance the classes on the Visual FoxPro desktop, instance the class on a live object, instance the class in the Form or Class Designer, or create instance code in the Command Window.

NOTE: All drag operations from the Class Browser are started by selecting the class in the TreeView and then dragging the class icon located in the upper left corner.

Developers can instance a class on the Visual FoxPro desktop by dragging-and-dropping the class icon onto the Visual FoxPro desktop. If you hold the Shift key during the drag-and-drop the class is created, but it is not visible. Holding the Ctrl key as you drag-and-drop the class will also display any errors that occur while the object is being created (see **Figure 2**). These errors are normally ignored when creating classes in this fashion. Classes are instanced as objects on the Visual FoxPro `_screen` object. The object references will be the class name with a numeric counter attached. Strangely, they cannot be seen in the Property Sheet, but can be seen in the debugger Watch window by adding `_screen` and are available to IntelliSense. If we dragged a class called `txtBase` to the Visual FoxPro desktop we can reference it in the Command Window using code like this:

```
?_screen.txtBase1.Name
_screen.txtBase1.Visible = .F.
_screen.RemoveObject("txtBase1")
```

This allows you to set properties and call methods to test the class live on the desktop.

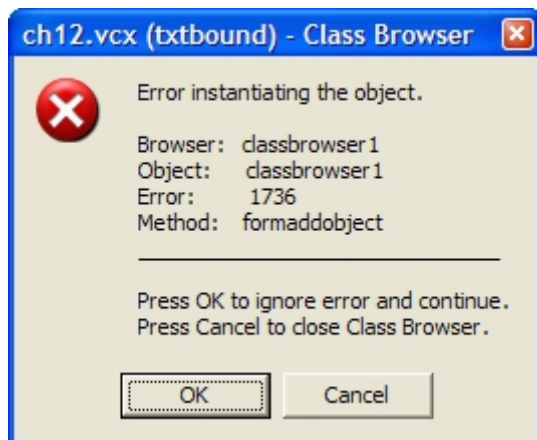


Figure 2. The Class Browser displays error information when holding the Ctrl key and dragging the class to the Visual FoxPro desktop.

If you drag-and-drop the class icon onto the Command Window Visual FoxPro will generate the `NEWOBJECT()` function call in the Command Window and instance the class:


```
oTxtbase1 = NEWOBJECT("txtbase","d:\data\winword\megafox\chapter12\ch12.vcx")
```

Subsequent instances of the same class are named the same with an incrementing numeric value attached to the end of the name. The class instances are directly available as memory variables and can be seen as instances in the Class Browser in the right side pane.

You can drag-and-drop classes onto a running form, any class that is a container, or to a form or class opened up in the designer in the same manner as described for the desktop or Command Window. To add an instance of the selected class to a live form programmatically you select the class in the Class Browser that you want added to the form. Put the mouse on the form where you want the class added and executed the following code in the Command Window:

```
_oBrowser.FormAddObject(SYS(1270))
```

Now you can test the class instanced on the form. If you are satisfied that it is working and you have the form instance with a variable called oFrmTest1, you can then save the live form with code similar to this:

```
oFrmTest1.SaveAs("MyCustom")    && Direct save to SCX  
oFrmTest1.SaveAsClass("MyClassLib.vcx", "frmTest2")    && Subclass of the class
```

Gotcha here is that the form or class cannot be saved to the file of the form or class currently instanced since it is already an object in memory. Therefore you will have to save it to a different form or class and work later to rename it.

Viewing and editing superclass code via the Class Browser

Most Visual FoxPro developers are familiar with a free tool called SuperClass. The SuperClass tool can do a number of things, but the feature it is most famous for is opening up the superclass method code of the method you are editing and allowing you to directly edit it without opening up the superclass first. This powerful tool was written by Ken Levy (who also happens to be the architect and developer the Class Browser back in the day when he was a contractor doing development for Microsoft on the Fox Team). This same feature is available if you are editing a class via the Class Browser.



Figure 3. The Edit ParentClass Method toolbar button is available if you edit a class from the Class Browser.

The Edit ParentClass Method button appears on the Visual FoxPro toolbar whenever a Class Browser window is active. This button allows you to view and edit the immediate parent class method while you're in the Form or Class Designer method editor. If you are not interested in this functionality you can disable this by setting a property of the Class Browser using the following code in the Command Window when the Class Browser is open:

```
_oBrowser.lParentClassBrowser = .F.
```

You will be prompted to save the superclass code after you close the window. The only disadvantage to editing code in this manner is that you lose editor colorization and the IntelliSense is not active.

Class Browser can open/maintain PRG based classes (VFP 9)

[\(CBPRGClasses.prg\)](#)

The Class Browser has always been able to open VCX files, but many developers prefer to code their classes in program code (PRG). One of the disadvantages to taking the PRG route was the inability to use the Class Browser to maintain the classes and get a visual representation of the class hierarchy. Visual FoxPro 9 removes this limitation.

Class Browser Add-in to retain the Regional Settings for time and date

(CbChangeDateFormat.prg)

We like when applications respect the Windows' Regional Settings for date and times. We write our applications to respect the user selections via the `SET SYSFORMATS ON` at the beginning of the applications. We noticed that the Class Browser does not respect the Regional Settings and defaults to `SET CENTURY OFF`. To combat this situation we wrote a simple add-in to the Class Browser that sets the date and time settings to the developer preference.

This Class Browser add-in can be run two ways. The first way is to just run the program from the Command Window. The add-in will recognize the fact that it is not being run from the Class Browser and will toggle into "registration mode". Registration mode adds a record to the Browser.dbf file. This table contains the registration information of all the add-ins. The Browser table is self explanatory and documented in the Visual FoxPro help file. The key points we want to make about the registration of this add-in is the Method column is "Activate" and the Program column is the name of the program being run. By setting the Method column to "Activate", we are telling the Class Browser to execute this add-in every time the Class Browser is activated. The registration code only needs to be run once on each development copy of Visual FoxPro you have loaded. If it is run more than once it will update the registration record.

Listing 1. Class Browser Add-in that changes date and times displayed in the Class Browser to the ones set up in the Windows' Regional Settings.

```
LPARAMETERS toBrowser

LOCAL lcName                && Name of the Add-in
LOCAL lcComment             && Comment for the Add-in
LOCAL lnOldSelect           && Save for reset later

* Self registration if not called form the Class Browser
IF TYPE("toBrowser") = "L"
    lcName      = "Rick Schummer's Date Setting Changer"
    lcComment   = "Developed by RAS for online forum discussion and example"

    IF TYPE("_oBrowser")= "O"
        * If Class Browser is running, use Addin() method
        _oBrowser.Addin(lcName, STRTRAN(SYS(16),".FXP",".PRG"), ;
            "ACTIVATE", , , lcComment)
    ELSE
        * Use the low level access of the Browser registration table
        IF FILE(HOME() + "BROWSER.DBF")
            lnOldSelect = SELECT()

            USE (HOME() + "BROWSER") IN 0 AGAIN SHARED ALIAS curRASDateChanger
            SELECT curRASDateChanger
            LOCATE FOR Type = "ADDIN" AND Name = lcName

            IF EOF()
                APPEND BLANK
            ENDIF

            * Always replace with the latest information
            REPLACE Platform WITH "WINDOWS", ;
                Type        WITH "ADDIN", ;
                Id          WITH "METHOD", ;
                Name        WITH lcName, ;
                Method      WITH "ACTIVATE", ;
                Program      WITH LOWER(STRTRAN(SYS(16), ".FXP", ".PRG")), ;
                Comment      WITH lcComment

            USE

            SELECT (lnOldSelect)
        ELSE
            MESSAGEBOX("Could not find the table " + HOME() + "BROWSER.DBF" + ", ;
                please make sure it exists.", ;
                0 + 48, ;
                _screen.Caption)
        ENDIF
    ENDIF
ENDIF
```

```

RETURN
ELSE
  * Check to see if we really got called from the Class Browser
  IF !PEMSTATUS(toBrowser, "lFileMode", 5)
    RETURN .F.
  ENDIF

  * Now the simple stuff to change the Date environment
  * setting which is specific to the private datasession.
  * This setting is driven from the developer's Windows'
  * Regional Settings.
  SET SYSFORMATS ON

  * Then refresh the Class Browser to reflect the change
  toBrowser.Refresh()
ENDIF

RETURN

```

If the code is executed by the Class Browser (remember that the Class Browser will automatically execute this code each time it is activated), it will have a reference to the Class Browser that ran this code. The Class Browser always passes a reference to itself when it executes an add-in. The code determines if the reference really is a Class Browser. If it is the `SET SYSFORMATS ON` is executed and the instance of the Class Browser is refreshed. The date and time at this point should be in the same format as the Windows' Regional Settings.

Project Manager

The Project Manager has been around for a long time, yet I am often impressed by the features that it has that it has to make me more productive.

Drag-n-Drop

Many developers are surprised to find out that the Project Manager is a drag-n-drop client and server. This means that files can be dragged to the Project Manager from many sources including other projects and from outside of Visual FoxPro. If you drag files from the Windows Explorer all the files will be added to the project.

Dragging from one project to another

Dragging files between two different projects creates a reference in the second project for that file. If there is a description for the file, this description is also added to the second project, even for files that don't store the description in the file itself. If the file is a program set as the main program of the originating project, VFP will prompt you with a question that asks if you want to make the file the main program in the second project. You do not need to be on the same page in each project. The file is naturally added to the correct category based on the file extension.

Drag objects from a project to a designer

Dragging files from the project to a form or class designer can save time during development. Many project objects can be dragged to the Form or Class Designer. The dropped objects are instantiated in the designer.

Dragging a field from a database contained table, view, or free table to a form or class will instantiate the associated class for the data type. The advantage of this feature is that it creates a bound object in the class without the use of a dataenvironment. Many developers we have instructed over the years feel they need to first drop on a class and then set the ControlSource.

While the manual setting of the ControlSource works, it requires developers to perform an extra step. The other advantage of this technique is that it incorporates the IntelliDrop capability seen when you perform this operation from the dataenvironment. This way the classes specified in advance are used instead of the VFP base classes. Tables dragged to a form or class will instantiate a grid. If you right-click and drag, you are presented with the

option of the grid class (or other class you have set for the Multiple setting in the Field Mapping in the Tool|Options).

If you want a specific class dropped on another container class, you can select it in the Project Manager and drag it to the container class. This does not bind the object like the drag operation of a table field. This allows you to override the IntelliDrop settings that are premapped. We expected that an icon dropped on a form might set the form Icon property and that dropping a graphic would generate an image object. They don't. Other objects not mentioned in this section cannot be dropped onto a form or container class.

Dragging from project to program code

In the same spirit described in the preceding section, you can drag and drop different project objects to code editors. The name of the object is displayed in the code window. For instance, if you drop a field name in the Command Window, you get the field name. Unfortunately you do not get the table.fieldname syntax. This works for every object type in the project except the stored procedure names. The only objects that carry over the file extension are the files in the "other" category.

What happens when dragging from Class Browser or Component Gallery to a project?

We tested a couple of other ideas with this capability by dragging files from the VFP Class Browser and its close cousin, the Component Gallery. We expected that dragging classes from the Class Browser to the Project Manager would add the class. Unfortunately, this does not work. We dragged the icon in the upper left corner to the project, just like we have many times to another instance of the Class Browser. It plain does not work in the VFP 6 or 7.

Even more interesting is that performing the same tests in the Component Gallery proved successful. Pick the class in the Component Gallery and drag the icon in the upper left hand corner or the icon in the right pane and drag it to the Project Manager. If the file is a class, there is a dialog presented which is asking how you want to add the file to the project (see **Figure 1**). You have an option to add the existing class library or add a copy of the class to another class library. The file is added. Cool! Any file that can be added to a project can be dropped from the Component Gallery. Quickly adding common files to a project from your favorite catalog is a pretty powerful technique!

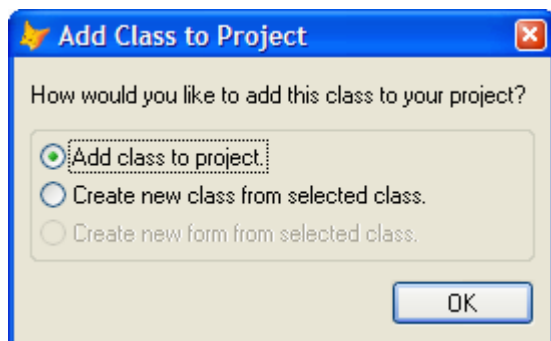


Figure 4. Adding a file to a project from the Component Gallery will prompt you to determine how you want the class added to the project.

Dragging from a non-VFP application to a project?

Another cool capability is that files can be dragged onto the Project Manager from outside of VFP. The Project Manager is just one of many parts of VFP that were enabled for OLE Drop and Drag in VFP 6.0.

So how can I leverage this feature? Pop open Windows Explorer or your favorite replacement. If we drag and drop VFP specific files from Explorer to the Project Manager, they are added to the appropriate category. Non-VFP files are added to the "other" category. If we drag a shortcut from the desktop, we get a shortcut (.LNK) file added to the "other" category in the Project Manager.

Open classes in Class Browser (VFP 9)

Double-clicking in the Project Manager on a Visual Class Library will open the class library in the Class Browser. Previous to VFP 9 nothing happened when you double-clicked on a class library. You can stop this from happening via a projecthook *QueryModifyFile* event. You can also intercept this in a projecthook's *QueryModifyFile* event and run your own class library tool:

```
LPARAMETERS toFile, tcClassName

#DEFINE IDYES          6      && Yes button pressed
#DEFINE IDNO           7      && No button pressed

* Only for class libraries
IF LOWER(JUSTEXT(toFile.Name)) = "vcx"
    * Only when no class is selected
    IF EMPTY(tcClassName)
        lnAnswer = IDNO

        * Offer choice if this option is enabled
        IF this.lofferChoiceOfClassBrowser
            lnAnswer = MESSAGEBOX("Would you prefer to open in VFP Class Browser even "+ ;
                                "though you have HackCX option enabled via projecthook?", ;
                                4+32, ;
                                "Modify Class Library")

        ENDIF

        IF lnAnswer = IDYES
            RETURN .T.
        ELSE
            * Run your favorite class library tool
            DO hackcx4.exe WITH toFile.Name
            RETURN .F.
        ENDIF
    ENDIF
ENDIF

RETURN
```

Build messages recorded immediately and to Debug Output window (VFP 9)

Previous to VFP 9, if a build stopped for any reason, the error log was not generated. VFP 9 generates the log as it finds errors so if the build stops you can still review the errors. If the Debug Output window is available, the same messages are output to this window (which you can save to a file if you want).

Change Font in Project Manager (VFP9)

While changing the font in a treeview is hardly a big bang to productivity, it is my opinion that anything that enhances usability and accessibility is an improvement in productivity for someone with a vision handicap.

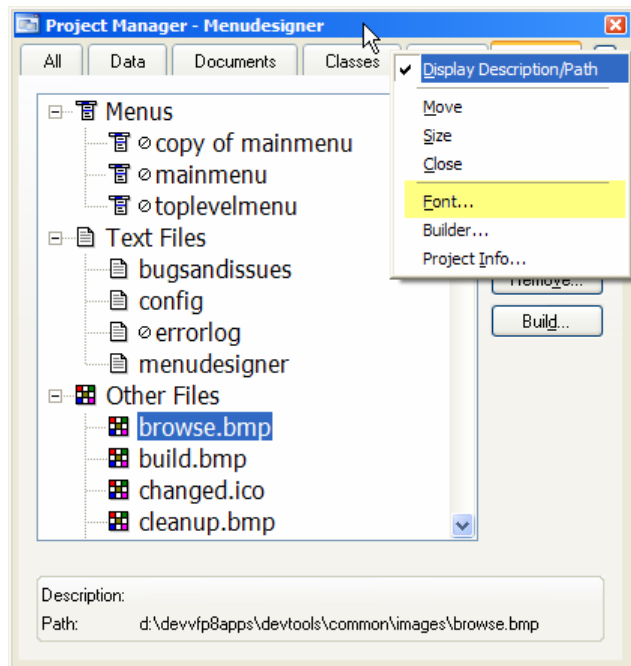


Figure 5. The Project Manager has the option to change the treeview font.

Task List

The Task List tool is new in Visual FoxPro 7. It provides a mechanism to track a list of to-do items and tasks within the development environment. Initially one might think this is not such a big deal with all the nice task tracking capabilities of your favorite personal digital assistants (PDAs) or applications like Outlook. The importance of the Visual FoxPro Task List lies in the integration with the various editors in Visual FoxPro. You can add bookmarks in the editors. These bookmarks translate into shortcut tasks in the Task List. You can use the Task List to then open up the appropriate editor with the method or program available.

There are three types of tasks that are tracked by the Task List tool. Shortcuts are specific references to a line of code that you want to return to or want quick access. User Defined Tasks are added through the Task List tool user interface. The "Other" tasks can only be added to the list programmatically. They all track the same information in the task table (referenced with the system variable called `_FoxTask`).

The Task List tool is really two separate items coupled together. The first is the Task List engine; the second is the user interface. Together they comprise the TaskList.app which is executed when you select the Task List option from the Tool menu.

The Fox Team at Microsoft has documented the task list engine for Visual FoxPro developers to use. The entire programming interface can be found in the Architecture.doc file that can be found in the VFP\Tools\XSource\VFPSrc\TaskList\Documentation folder. This folder and the rest of the source code for the Task List (and the other Visual FoxPro Xbase tools) is available if you unzip the XSource.Zip file installed in the Tools\XSource folder.

Adding my own custom fields to the Task List?

(TLExt.dbf/fpt)

The Task List tool provides developers the capability to extend the data that is tracked for a task via custom fields. These fields are stored in a separate table with a one to one relationship to the main task list table. To access the field extension, right-click on the Task List and choose Options. There you can specify or create the "user-defined column table"

This custom field table must include a 10-character UniqueID field that is used to link the custom fields to the base fields. The UniqueID field is automatically included when you create a new table. All other fields are fair game. One recommendation we can make is to not name a custom field the same as one of the base fields (see **Table 1** for a complete list) to avoid confusion. Our testing revealed that all the Visual FoxPro data types are accessible from the Task List tool except for general, currency and the two binary fields for general and memo. If you add one of the banished data types the Task List engine will just ignore them.

Table 1. The base table contains the eleven fields for the developer to expose in the Task List user interface.

Field Name	Data Type	Size
UNIQUEID	Character	10
TIMESTAMP	Numeric	10
FILENAME	Memo	4
CLASS	Memo	4
METHOD	Memo	4
LINE	Numeric	6
CONTENTS	Memo	4
TYPE	Character	1
DUEDATE	Date	8
PRIORITY	Numeric	1
STATUS	Numeric	1

The Task List Options dialog does not provide a mechanism to add additional fields to the current custom fields table. You can take control of this situation using the trusty **MODIFY STRUCTURE** command and make the appropriate changes. The Task List tool will recognize the changes the next time it is started.

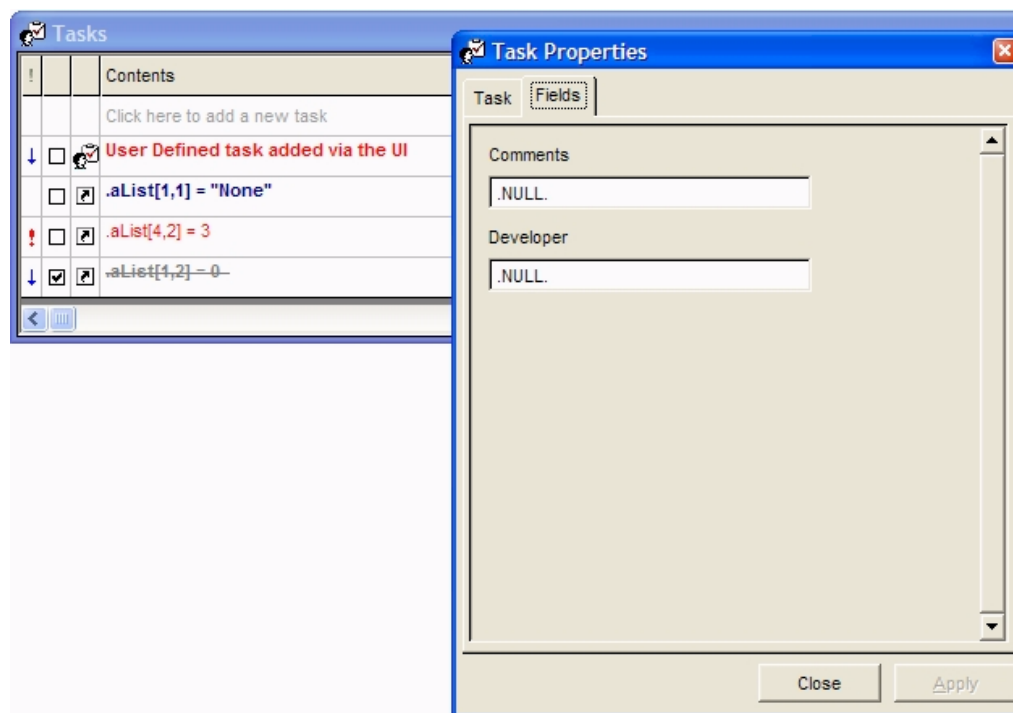


Figure 6. The Task Properties form is not very helpful in displaying the values stored in the custom properties.

There are a number of extended fields we have added to our user-defined fields including developer (to differentiate team tasks), comments or instructions (to allow developers to type in comments about the task or instructions for other developers or a reminder for yourself), and project.

The problem with these fields is that there is no hook in the Task List to provide default values, and since they are free tables, no mechanism to hook into the database engine. Later in this white paper we will discuss a solution to this issue.

Using my custom fields in the Visual FoxPro Task List

Once you have established your own custom fields for the Task List you need to add the columns to the user interface to be able to fill in the information, or open the task dialog and check out the Fields page.

The Task List context menu has a Column Chooser option. Selecting this will bring up the Column Chooser dialog. The base fields and the user-defined fields are available to be added to the user interface. Only the fields not currently in the user interface are available. The position of the column will depend on the current active column. Select the column you want the new column to be added next to (to the right of the column selected).

Opening up the task (also started from the context menu) brings up the Task Properties, which has a second page that displays all the custom user-defined fields. We have found that this page is broken from a display perspective. All the fields default to the NULL value. Another thing is lacking is that all the fields are exposed in textboxes, even for memo fields. The page allows data entry and saves the changes, but the changes do not get displayed the next time the page is displayed. If the column is in the Task List you can see the change made in the Task Properties dialog.

Fixing the Task List when it seems to have lost its mind

Occasionally the Task List will lose its mind. We have to remember that the Task List tool in VFP 7 is a 1.0 release, and like most software with that version, there are quirks and bugs. Some of these bugs will disable the Task List tool from even starting and can easily make it unusable. We have a technique that restores some stability to a Task List that will not start, might not show all the tasks in the FoxTask.dbf, or might just be crashing on loTask variable not found errors.

The tasks for the Task List are stored in a table called FoxTask.dbf. The table used by the current Task List is stored in a VFP system variable called **_FoxTask**. You can open this table via a standard USE command and manipulate the records. We have found that adding and deleting records to this table without the task list engine reference can be troublesome to the user interface. We have noticed that modifying the existing records for the due date and contents is not usually a problem, but have established some instability when making changes to the other base fields.

The configuration items that define some attributes of the task list engine and the user interface are stored in the FoxUser.dbf resource table. There are five records in the resource table. Deleting the five records will force VFP to recreate the default settings and a new set of records in the resource table the next time the Task List is restarted. To delete the Task List records use the following code:

```
SET RESOURCE OFF
USE (SYS(2005)) EXCLUSIVE
SET FILTER TO "TASK" $ Id
BROWSE
```

Delete the five records and then:

```
PACK
USE
SET RESOURCE ON
```

Restart the Task List tool. You will need to reselect the User-Defined Column Table via the Options dialog to re-establish your extended columns and then reset the columns displayed via the Column Chooser dialog.

Type	Id	Name	Readonly	Ckval	Data	Updated
PREFW	TASKCOLS	memo	F	34489	Memo	02/24/2002
PREFW	TASKFORM	memo	F	63952	Memo	02/24/2002
PREFW	TASKCELL	memo	F	29490	Memo	02/24/2002
PREFW	TASKORD	memo	F	9003	Memo	02/24/2002
PREFW	TASKEDIT	memo	F	12719	Memo	02/24/2002

Figure 7. There are five records in the FoxUser.dbf that retain the settings and configuration of the Task List.

Task List tasks disappear after I add an existing user-defined fields table

One Task List feature available to the developer is being able to select an existing user-defined column table. There is one problem with this; all the tasks in the Task List disappear. They are not deleted from the FoxTask table, they just do not show up in the user interface. The problem is that the Task List establishes a one-to-one relationship between the base fields table (FoxTask) and your custom fields table. This relationship will likely not be correct with an existing table. The solution is to manually open both the FoxTask table and the user-defined table and add the needed records to the user-defined table.

Putting it all together with the WLC Task List Editor

(Example: [TLEditor.scx/sct](#))

Microsoft has put in place enough hooks to completely build our own Task List Editor. So we decided to put some of this new found knowledge to work in a tool called the WLC Task List Editor and address some of the limitations of the Visual FoxPro Task List.

You can add (user defined and other tasks), update, and delete tasks. We decided not to add support for shortcuts task types since it is much easier to add tasks from the various code editors. The data is “record buffered” since all the data is accessed via the task object data. It can be reverted until you move to another task. You can accomplish the same thing in the Visual FoxPro Task List if you edit the data in the Task Property dialog, but the “grid view” of the tasks has no way to revert any changes.

There are a couple of features that are included in the WLC version that are not included in the one that ships with Visual FoxPro. The first issue addressed is that the native Task List does not display the existing information in the user-defined columns in the Task Properties. The example ships with _Developer and _Comments extended properties exposed. This version of the tool also provides a mechanism to have default values for the user-defined columns in the various add methods. The native tool does not have a mechanism to add “other” task types, this can be accomplished by clicking on the “Add Other” task button.

If you want to expose the custom Developer and Comment properties (extended fields exposed on the WLC Task List Editor) you need to add these user-defined columns to the Task List. If they are not in your list of custom field these properties are not exposed on the user interface. See section “How do I add my own custom fields to the Task List?” in this chapter with steps to add your own custom fields. These fields are also available in the sample extended column table supplied in chapter downloads. See TLExt.dbf/fpt.

There is a known issue with the initial release. If you delete all the tasks there are a number of problems with the WLC Task List Editor. It is causing the Content column of the grid to be truncated. Restarting the form will clear the problem. We are going to address this in a service pack. Check the Hentzenwerke website for updates.

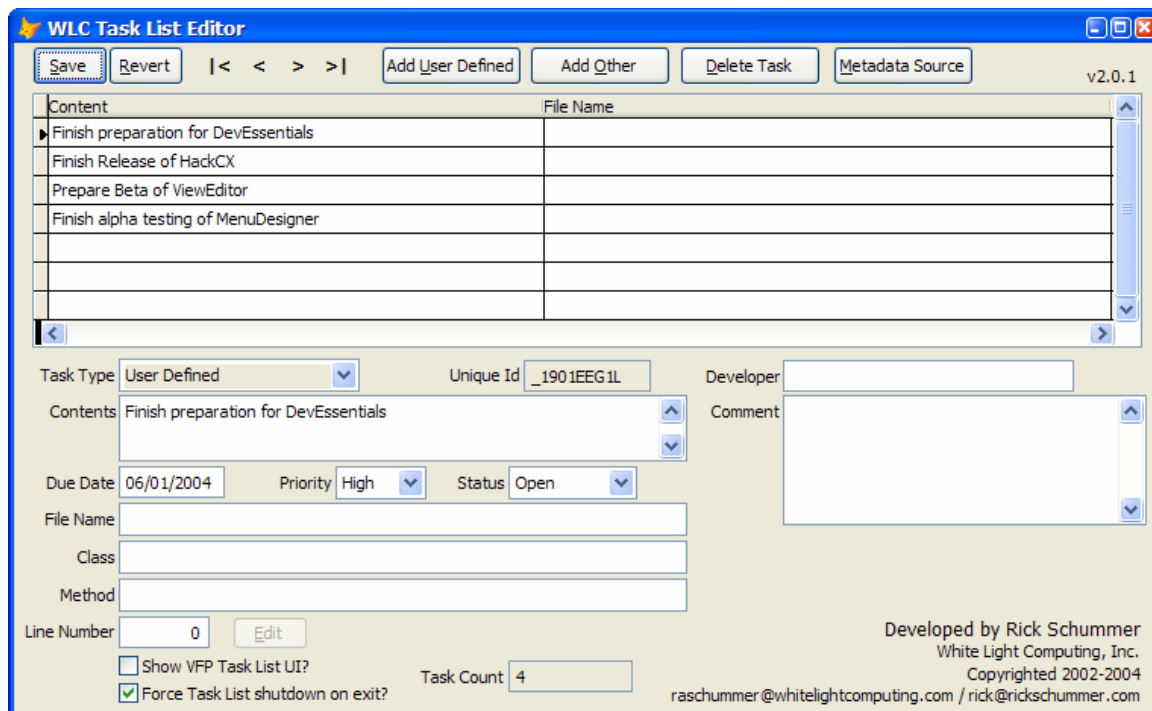


Figure 8. The WLC Task List Editor provides some functionality not available with the native Visual FoxPro Task List.

Future enhancements to this tool include capabilities to pack the metadata tables, filtering specific task types, recalling deleted tasks, copying a list of tasks to the clipboard (so they can be copied to an email, and generating some paper reports.)

Object Browser

The Object Browser is new with Visual FoxPro 7. It exposes the public and protected interfaces of COM object libraries and ActiveX controls. Inside these libraries is a wealth of information concerning the properties, events, methods, constant values, and classes available for developers. This tool is very important to developers who write Automation code and need to understand the documented ways of using a particular Automation object.

Determining the values of constants defined in a COM object

One of the truly grueling tasks developing automation code is determining the constants used in the examples. These constants can be translated into **#DEFINE** code. Before we had the Visual FoxPro Object Browser we needed to trudge through help files, hope examples documented the values, or use a tool like the object browser found in the VBA editors of Microsoft Office to find these values. This was a time consuming process for sure. Tools like the West Wind GETCONSTANTS.EXE would read the type libraries and generate the **#DEFINE** code which is easily compiled by Visual FoxPro.

The Object Browser can generate the **#DEFINE** code efficiently and is a real time saver. To accomplish this, open up the COM or ActiveX component, drill down the TreeView to expose the Constants node. Open up a program editor (program, or class method). Drag the Constant branch and drop it in the editor. Not only is the **#DEFINE** code typed in with the constant name and value, but the documentation for the constant is also included as a comment for the **#DEFINE** if the constant has a description. If you drag the constants branch you will get all the constants in the editor. You can also drag individual constants if you only need specific ones.

GOTCHA: We have experienced constants that are decimal values will get rounded to zero. If this is the case, we recommend the GetConstants.exe from West Wind, which does not suffer from the same bug.

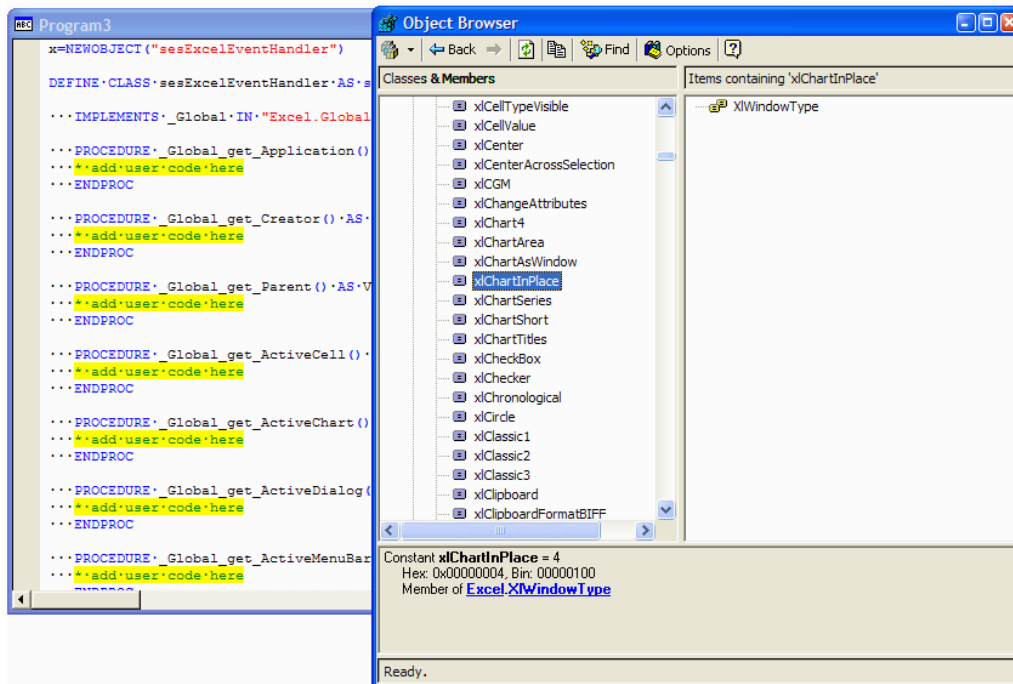


Figure 9. The Visual FoxPro Object Browser quickly generates `#DEFINES` for constants and the template code for a class that implements event handling functionality.

Use the Object Browser to create class templates to implement interfaces

A very powerful new feature in Visual FoxPro is the capability to write code in our custom applications that respond to events in other applications. For instance, we can now write code to respond to a user closing a spreadsheet, or sending an email in Outlook, or doing a mail merge in Word. This is done with the new **IMPLEMENTS** clause of **DEFINE CLASS** as well as the new **EventHandler()** function.


The Object Browser assists developers in writing tedious code in this respect. First open up the COM or ActiveX control in the Object Browser. Then drill down through the TreeView and locate the Interfaces node. Open up a program editor (program, or class method). Drag the interface node and drop it in the editor. The class definition is written, including the **IMPLEMENTS** and template code for each of the methods that are exposed. All you have to do at this point is rename the class from `MyClass` to something more descriptive, and add code to the appropriate method.

Find out the name of the OCX file to ship with my deployment setup

The new Object Browser helps Visual FoxPro developers with numerous features for ActiveX controls. One of the simpler, yet more helpful items is that it displays the actual file name for the OCX and other details about the control.

Open up the Object Browser and select an ActiveX control from the list. If you select the root node for the control there is details about the OCX displayed in the bottom pane of the Object Browser. Information like the file name, the help file, and the GUID is presented for the developer. This can come in handy when you need to find out what OCX file is to be included in a deployment package, and determine where the help file is installed on the hard drive.

Task Pane

The Task Pane is a new tool that ships with VFP 8. It is a browser based interface that runs inside a Visual FoxPro form. The Task Pane is available on the Tools menu or via the Standard toolbar. 

Add shortcuts to your developer tools

The Start page of the Task Pane has several sections. The second section allows you to add developer tools. These tools can be tools that you develop, other developers have developed, or ones provided with Visual FoxPro.

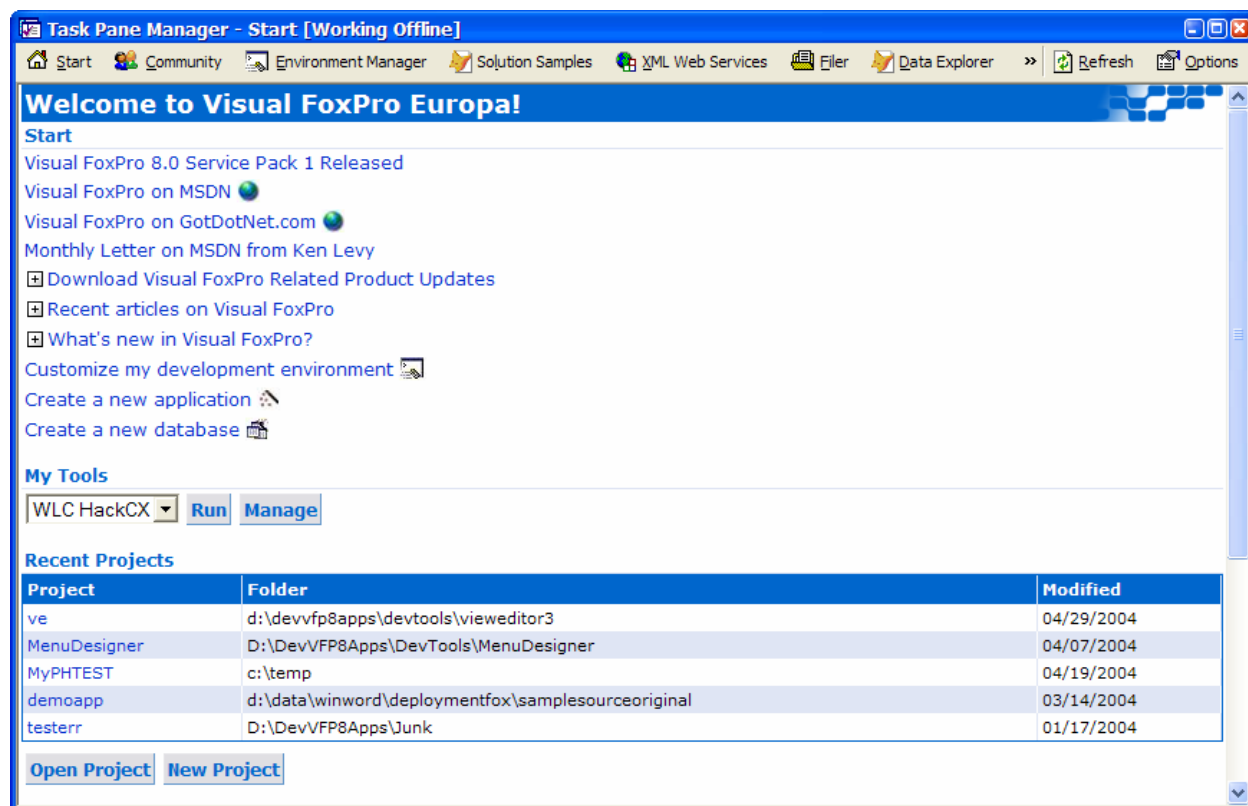


Figure 10. The Visual FoxPro Task Pane allows developers to hook in developer tools.

The shortcuts can be to VFP executables (APP or EXE), forms, or reports. If this is not what you are looking for, you can write script (Visual FoxPro code) to do anything you can do in Visual FoxPro. A trivial example of this might be to get a file name to the Windows Clipboard:

```
_cliptext = GETFILE()
```

Shortcuts to key VFP news sites

Microsoft and the Fox Team in particular is very developer community centered. They have provided a link to many popular Visual FoxPro centric Web sites and to the two major news sites. You can determine which sites you want the news from, and how many days to look back.

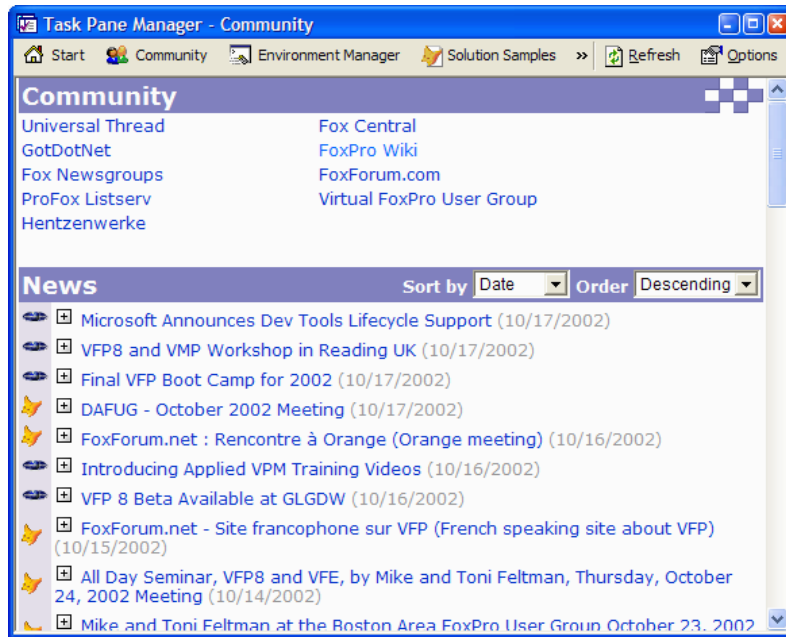


Figure 11. The Visual FoxPro Task Pane provides links to the news items on Universal Thread and FoxCentral.

At this time the Community section listing of Web sites is not configurable.

Managing the VFP environment

There are many environment settings that developers typically set up in the startup program that is run when Visual FoxPro begins. Some developers have sophisticated projecthooks that configure the development environment settings each time a project is opened or activated. What I am referring to is the some of the many **SET** commands available (Talk, Path, Deleted, Exclusive). The Task Pane provides developers that are not using either of these techniques to create environment states that are saved and can be loaded to set the various environment settings.

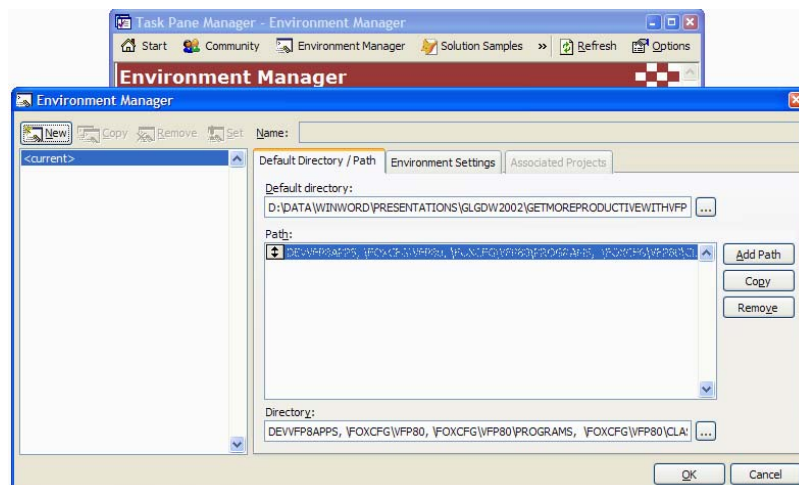


Figure 12. The Visual FoxPro Task Pane has the ability to have environment configurations that you can switch between as you are developing different projects or even within the same project.

Field Mapping in Environment Manager (VFP 9)

The Field Mapping dialog in the VFP Tool Options dialog has to be the worst dialog in all of Visual FoxPro. It is tedious to use and very unfriendly. If you work on multiple projects with multiple frameworks or even just different base classes for the project, then the new Field Mapping capability of the Task Pane Environment Manager introduced in VFP 9 is the ticket for you.

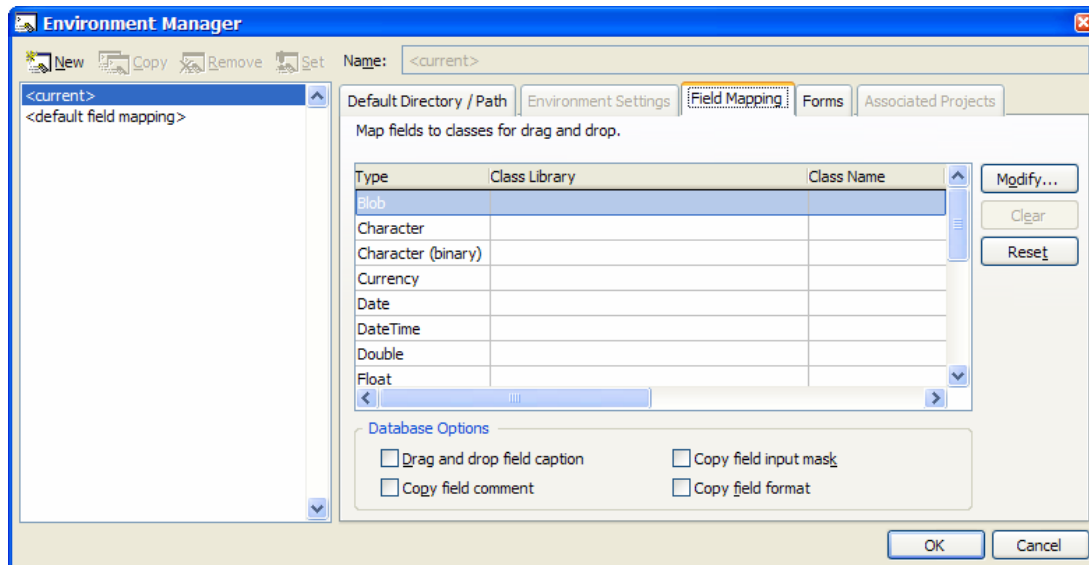


Figure 13. The new Field Mapping in the Environment Manager can save a lot of time if your different projects have different base classes.

SQL Data Explorer (VFP 9)

The new SQL Server Data Explorer has a number of features to query data in any of the databases available on any of the database servers. It is a watered-down version of SQL Server's Enterprise Manager. The thing we find productive is the fewer clicks to browse the record set. Unfortunately all the data is read-only

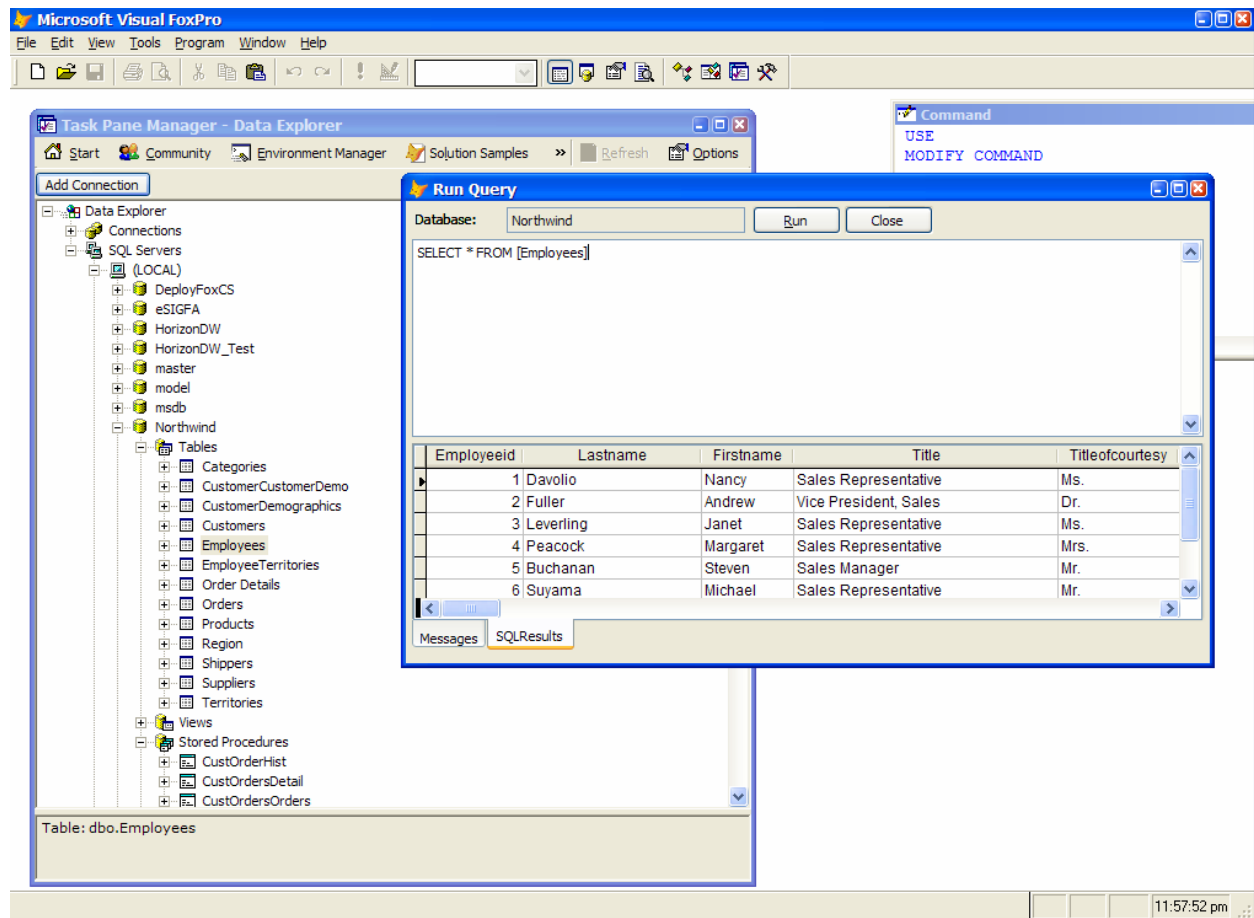


Figure 14. The Data Explorer allows you to query SQL Server data.

Exploring Web services

Visual FoxPro has been able to work with Web Services since version 7. The way to register Web Services is to use the IntelliSense Manager. Once a Web Service is registered, Visual FoxPro has a number of ways to utilize the registered information. One of the ways it is utilized is to let IntelliSense help write code to use the Web Service. The Task Pane tool will provide an inside look into the Web Service and expose the public methods and provide code examples to use the methods once the reference to the Web Service is obtained.

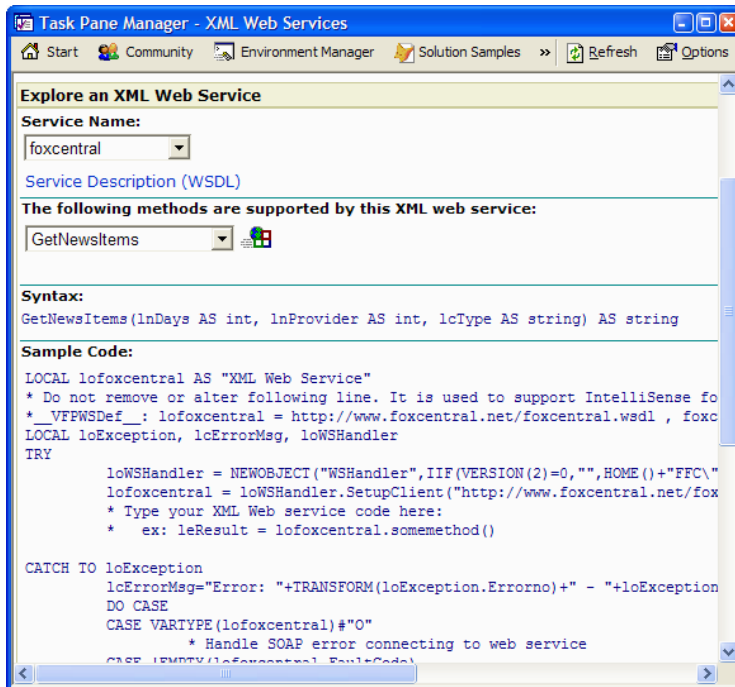


Figure 15. The Visual FoxPro Task Pane explores documentation of a webservice.

Fully Configurable

The Task Pane tool has a number of configuration settings available for Visual FoxPro developers to set for their own needs. I encourage you to explore the options and set them to your liking.

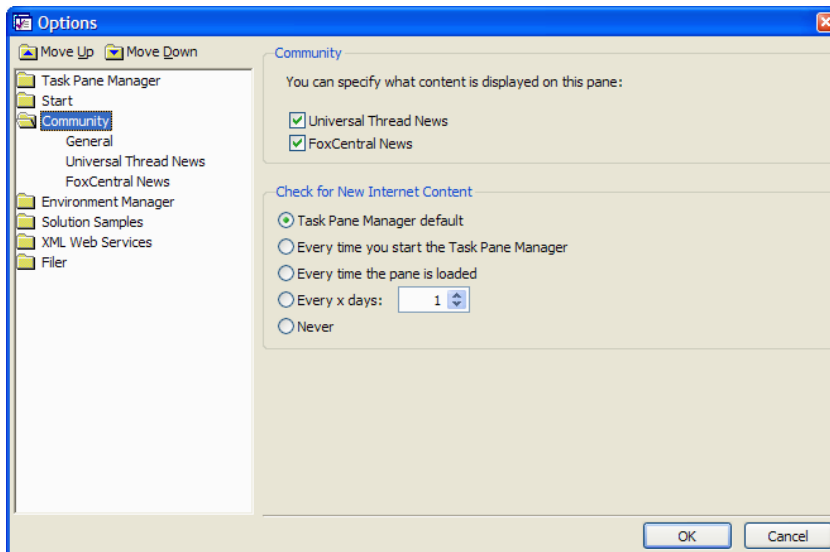


Figure 16. The Visual FoxPro Task Pane is fully configurable via the Task Pane Options dialog.

Toolbox

The Toolbox is new in VFP 8 and is designed to provide a Form Controls toolbar on steroids. The Form Controls toolbar provides developers a single class library, the base classes, or the ActiveX controls selected in the VFP

Option dialog. The ToolBox provides groupings of classes, ActiveX controls, and the ability to write scripts from text blocks.

Text Blocks

Text blocks allow you to drop and drag text into a program editor. If you mark the text as **TEXTMERGE** evaluation text, you can literally write VFP **TEXTMERGE** script.

ActiveX Controls

Dragging the ActiveX controls to an editor provides the needed **CREATEOBJECT()** code. You can easily change this into a **LOCALS** declaration statement to enable IntelliSense in the editor. The following code was created when I dropped the DynaZip Zip ActiveX Control in the editor:

```
Olecontrol = NEWOBJECT("dzactxctrl.dzactxctrl.1", "dzactx.dll")
```

This can be changed to:

```
LOCAL loZip AS "dzactxctrl.dzactxctrl.1"
```

Dragging the ActiveX controls to the Form or Class Designer will instantiate the controls on the form or class.

Dockable (VFP 9)

The Toolbox is now a dockable form. This is not productive by itself, but the fact that I no longer need to move windows from underneath the Toolbox does save me time.

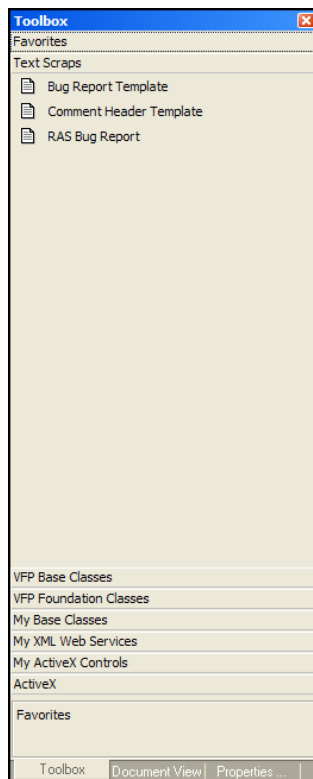


Figure 17. The Toolbox can be docked individually or as part of a multiform dock.

Code Reference

The new Code Reference tool is designed to do project and folder wide search and replace (optional) for text. This is a very, very nice addition to Visual FoxPro. The tool surpasses other tools that have been developed by the Fox Community.

From a productivity view, the search saves time looking for text in each individual form, program, menu, etc. The results of the search are saved so they can be reviewed later so you do not have to rerun searches. The existing saved searches can be refreshed (rerun) with a click of a button which saves you time re-entering the search criteria. The results can be printed to create a check list for you or developers on your staff. The tool allows you to export the results to any of the `copy` to formats with the capability of selecting the records output.

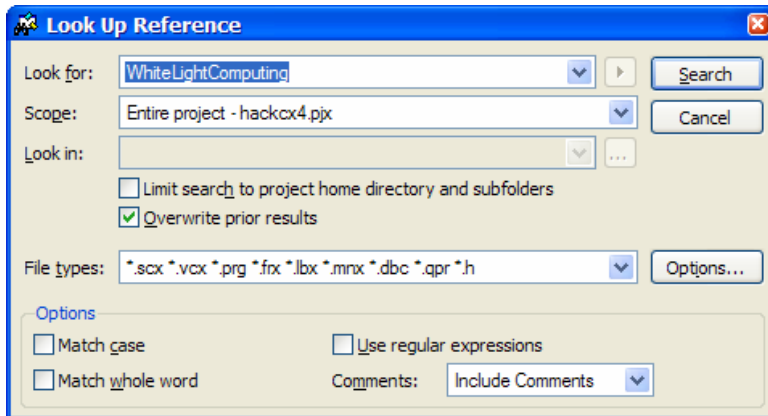


Figure 18. The Visual FoxPro Code Reference Look Up Reference provides access to the text search criteria.

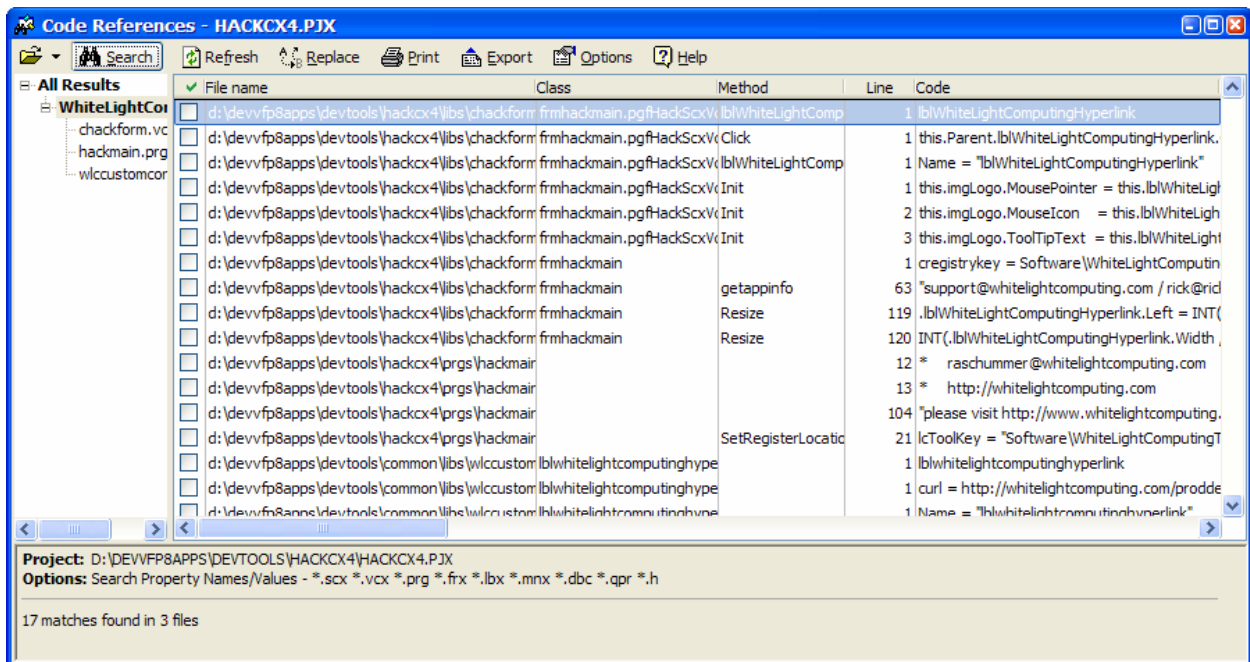


Figure 19. The Visual FoxPro Code Reference displays the results of searching text.

The replaces perform a potentially destructive operation, but it also creates backup files so you can retrieve the previous state. We still would recommend developers making copious backups of your source code.

Visual FoxPro 9 improves the Code Reference tool by providing new sort options and breaking up the Class/Method/Line column in the grid. You need to choose the different behavior by going into the options and selecting the separate columns. The results treeview has new options on the shortcut menu as well. You can not expand all nodes, collapse all nodes, and sort the most recent first. These options are all productivity increases if you have a lot of results from your searches.

Even when it was shipped in VFP 8 as a v1.0 tool, this one is very good. The version shipping with VFP 9 has been improved. There are developer community based tools available like Steve Dingle's Project Searcher (available at www.stevedingle.com)

Editors

Visual FoxPro developers spend an enormous amount of time in the various code editors (programs, methods, stored procedures) so we have a number of productivity tips for you.

Document View

The Document View was added in Visual FoxPro 7 to improve navigation of programs with numerous procedures, functions, methods, #DEFINES, and preprocessor directives (#INCLUDE, #IF, #IFDEF, and #IFDEFs). Each of these are like a bookmark entry in the code. You can sort the list by name, locations, or types. You can also control the font used in the window to make it smaller to make more items visible, or bump up the font if you want to see the entries.

See Bookmark section below to generate dynamic points of entry into your code.

Beautify Code Blocks (VFP 8)

The Beautify tool can be used to work on a block of code, not just the entire editor worth of code. You might be use to selecting the Beautify option from the Tools menu, but in VFP 8, Microsoft has added this option back on the shortcut menu.

Printing Source Code Files and Syntax in Color (VFP 8)

Source code can be printed in color as long as you have a color printer and the output is set to color in the Print Properties dialog (set from the Print dialog). Colorized source includes any of the color syntax capable editors (programs, methods, stored procedures or memos). Colors printed will match the appearance of the code syntax color and are based on your preferences.

Pasting code retains colors (VFP 9)

You will see code examples in this whitepaper that are black and white. These code examples were pasted in this Word document from VFP 7 and VFP 8. There are also "colorized" examples in this document. These examples were cut and pasted from VFP 9. The Fox Team saw examples of code posted on forums in color thanks to some Web services that developers have created. What they have done is added new information on the clipboard so tools like Word can paste more information into the document. We find this very cool and productive because the code is colorized exactly how we have set up the syntax forecolors for the Visual FoxPro editors. Here is another example of colorized code:

```
* RAS 17-Oct-2002, created for GLGDW 2002,  
*   updated for DevEssentials 2004  
*   updated for Southwest Fox 2004  
  
* Setting default class library, with  
* class open in Class Browser  
_oBrowser.SetDefaultFile()  
_oBrowser.ResetDefaultFile()
```

```

* Programmatically open class libraries
DO (_browser) WITH "examples\CPhkBase2"

* Open class, then position to specific method in class
DO (_browser) WITH "examples\CPhkBase2", ;
    "phkDevelopment.InstanceRegistry"

* Open class, then position to specific property in class
DO (_browser) WITH "examples\CPhkBase2, examples\demo", ;
    "phkDevelopment.lCopyAppToTestDirectory"

* Open multiple classes
DO (_browser) WITH "examples\CPhkBase2, examples\demo"

* Open PRG-based classes (VFP 9)
DO (_browser) WITH "examples\CBPrgClasses.prg"

* Start the Class Browser with the "retro" look
DO (_BROWSER) WITH "", "", .T.

* Start the Class Browser maximized
DO (_BROWSER) WITH "", "", .F., "", 2

* Start the Component Gallery
DO (_BROWSER) WITH "", "", .F., "", 0, .T.

```

Background compilation (VFP 9)

Background compiles happen as you enter a line of code in the Command Window or one of the Visual FoxPro Editors. If you have used Microsoft Visual Studio .NET you might be use to the background compiler since the language editors compile the code “on-the-fly”. In .NET you get colorized squiggly lines under offending code. In Visual FoxPro you can configure how the background compiler flags offending code. The choices are:

- Red Inversion
- Grey
- Underlined
- None

The option is set on the Editor page of the Tools | Options dialog. Our testing during the beta, it is our perception that the background compile (either alone or in conjunction with the IntelliSense Engine) slows the interaction with the editor. Best case for performance was the underline option. We are hoping this gets addressed before the final version or we might just have to shut off the option.

As far as productivity, we see some huge advantages to developers new to Visual FoxPro because their code immediately gets flagged if something is not syntactically correct. Experienced developers might not gain that same productivity because they compile in their own mind as they are writing the code.

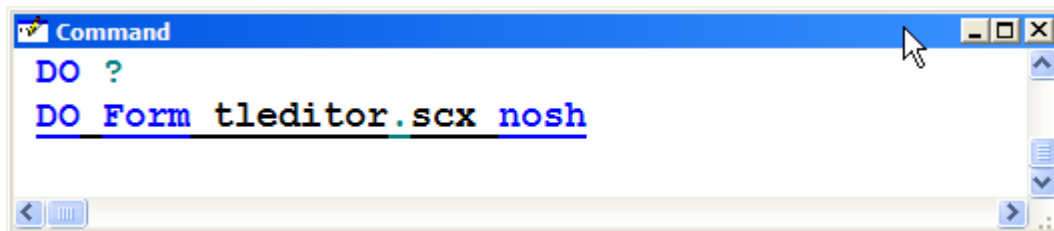


Figure 20. The Visual FoxPro 9 background compiler will flag code that is not correct syntax. This instance of Visual FoxPro has the setting configured to “underline”.

Select printed text (VFP 9)

You can now print just the selected code in an editor. This is handled in the updated print dialog.

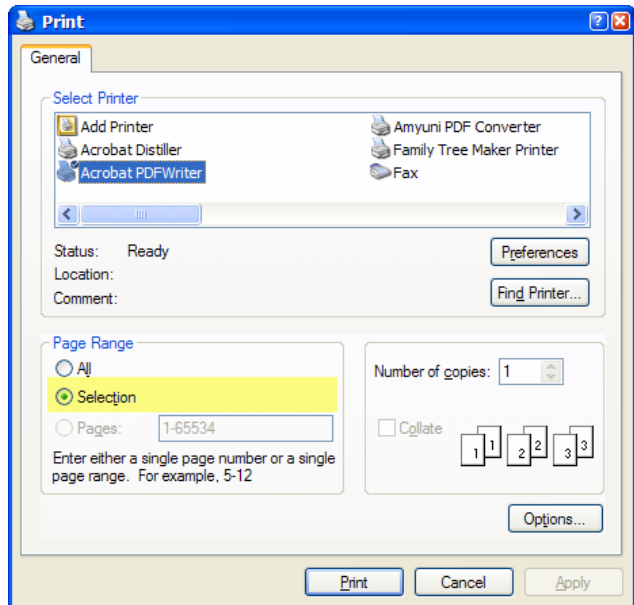


Figure 21. The updated Print dialog has the “Selection” option set if you have selected code in the editor and then use File | Print to print the code out.

SET PATH ... ADDITIVE (VFP 9)

(Example: [SetPathAdditive.prg](#))

How many times have you written code like this:

```
lcOldPath = SET("PATH")
SET PATH TO &lcOldPath ; Reports, Forms
```

Well how about:

```
SET PATH TO Reports ADDITIVE
SET PATH TO Forms ADDITIVE
```

You no longer need to save and reset the path, or go into the Tools | Options dialog to add a new folder to the list. A simple command in the Command Window and the path is extended.

Bookmarks

Bookmarks were introduced with the release of VFP 7. Bookmarks allow you to move easily to specific locations in your code in the various editors. You can set bookmarks and shortcuts, and then use shortcut keys to cycle through them. Bookmarks are temporary. They are retained when the editing window is closed and reopened, but they are not retained after exiting and restarting Visual FoxPro. Shortcuts are persistent between editor and VFP sessions. Shortcuts are stored in the Visual FoxPro TaskList (see Task List section earlier in this white paper).

Creating/Removing Bookmarks (toggle)

1. In the Editor, place the mouse cursor in the Selection Margin next to the code line you want to mark, and then press the Shift key and double-click.

2. In the Editor, place the cursor in the code line you want to mark, and then press Alt+Shift+F2.

Creating/Removing Shortcuts (toggle)

1. In the Editor, place the mouse cursor in the Selection Margin next to the code line you want to mark, and then press the Ctrl key and double-click.
2. In the Editor, place the cursor in the code line you want to mark, and then press Alt+F2.

Moving between shortcuts

1. Next bookmark, press F2
2. Previous bookmark, press Shift+F2

Gotchas

Bookmarks go away after you exit Visual FoxPro (fact, not a gotcha). Shortcuts are persistent, but the fact is that they are tied to the line number that they are created on, not the content of the line. This means that bookmarks created will not shift up or down as lines are added or deleted before the shortcut (or bookmark).

Property Sheet

I spend a lot of time in the property sheet so I love when Microsoft adds productivity to this feature.

Hotkey to locate property/method

The property sheet has a long list of properties you have to scroll down if you want to get to a specific one. Or do you? Scrolling up and down the list can consume lots of precious time. What if there was a shortcut? How about pressing Ctrl+Alt+<letter>? The first member (property, event method, or method) in the list on the current page will be displayed. Continuing the combination will take you to the next member matching your letter. When you hit the bottom of the list for the letter, the first member starting with the letter is selected.

Zoom window

The Zoom dialog provides you much more space than the cramped work area of the property sheet. This comes in handy for expressions that are long. VFP 9 increases the size of the Zoom window and the Fox Team added a new button next to the Expression Builder button on the property sheet. This was previously only available on the property sheet shortcut menu.

Member Data (VFP 9)

Member data is absolutely my favorite new feature in Visual FoxPro 9. Member data solves several enhancement requests Visual FoxPro developers have asked for a long time:

- Custom members (properties, methods) to have mixed case in the Property Sheet
- Custom member's mixed case to be respected by IntelliSense

In addition, the Fox Team in their usual way has extended this to the next level. They have provided developers to control exactly how properties are displayed, handled by IntelliSense, and if they are added to the new Favorites tab on the property sheet. You can set this for properties at the class level, or globally. For instance, you may want to have the Caption property always show up on the Favorites tab. No sense in adding this to the member data for all classes, just define it once.

To get to the basics, you need to look at a new property to the class called `_memberdata`. This property needs some XML to define how members for this class are handled. In our example we have a custom property called

ISecurityEnabled. This was added to this class. We also added the _memberdata to the class. Here is the XML string stored in the _memberdata property:

```
<?xml version="1.0" encoding="Windows-1252" standalone="yes"?>
<VFPData>
  <memberdata name="lsecurityenabled" type="property" display="lSecurityEnabled"
favorites="True" override="False"/>
</VFPData>
```

The XML stored in the _memberdata property is reflected in the property sheet. Note the mixed-cased settings for the property. This is determined by the display="lSecurityEnabled".

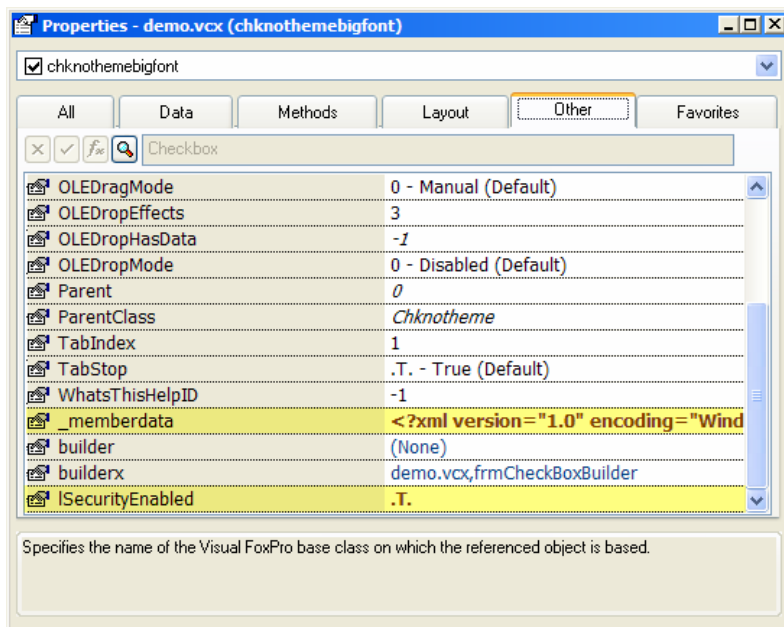


Figure 22. The property sheet reflects the mixed case set up in _memberdata.

Doug Hennig wrote a little tool to help get you started on the possibilities with _memberdata and has an article on this topic in the June 2004 issue of FoxTalk 2.0 which will give you all the details of how this is implemented using the IntelliSense metadata. His discussion includes a demonstration on how you can extend this architecture to create “property builders”. This tool is now a builder included in the base product.

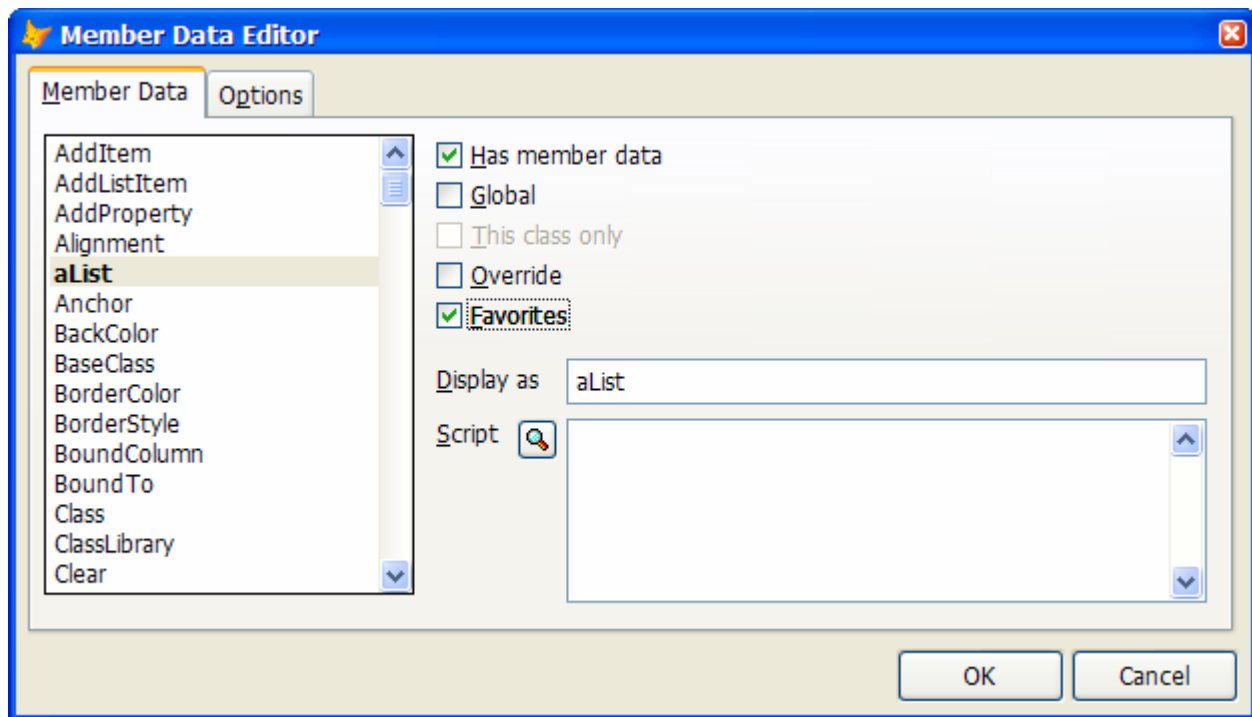


Figure 23. The Visual FoxPro Member Data Editor builder is a sure-fire way to simplify the editing of `_memberdata`.

The same mixed case is displayed in the editor through IntelliSense.

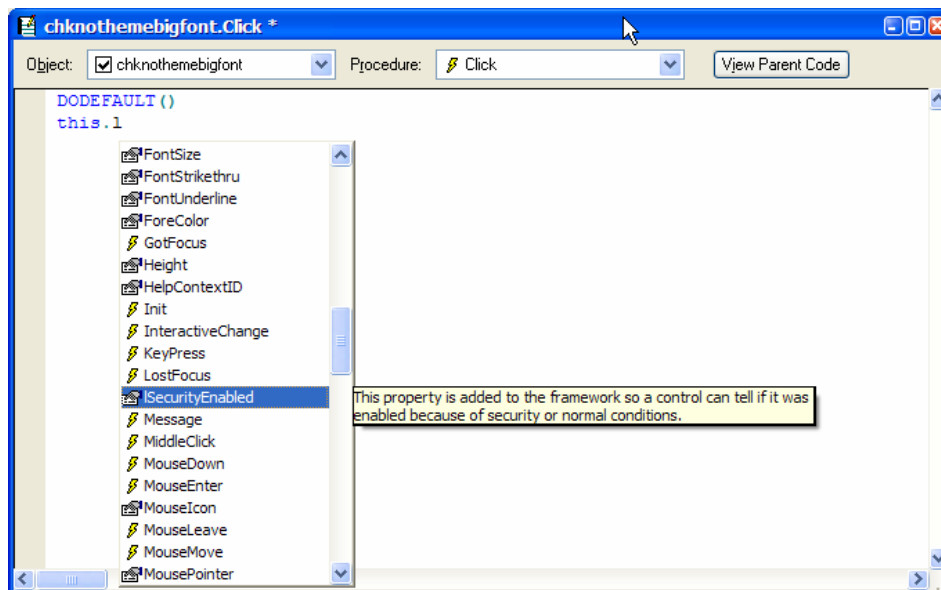


Figure 24. IntelliSense is even more productive with the inclusion of the mixed case members.

Favorites (VFP 9)

As I noted in the last section on member data, the metadata defined for member data allows you to define which members show up on the new VFP 9 Favorites tab on the property sheet. I think it is super productive to have the

most common or favorite properties and methods on the Favorites tab. It keeps me from jumping from one tab to the next looking and searching for the same property I go to every time I open this form or class.

Default values for New Property (VFP 9)

One of the features I added to HackCX Professional years ago is the ability to define the default property value when I add a new property. Microsoft added this to VFP 9 so we no longer need to perform the extra step of going to the property sheet, clicking on the Other tab and locating the new property. It is included in the New Property dialog.

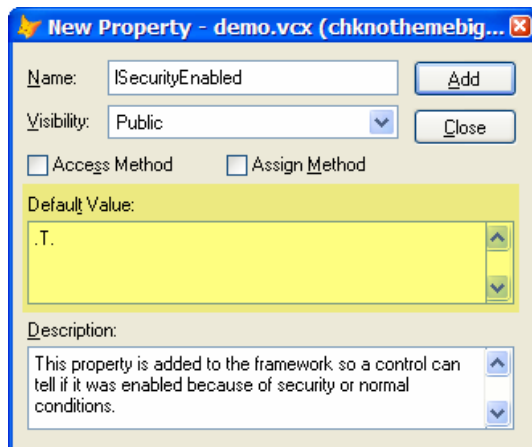


Figure 25. The New Property dialog allows you to preset the value of the property.

Font selection (VFP 9)

Previous to Visual FoxPro 9 you had three font size options with the property sheet: small, medium, and large. The Fox Team responded to our requests and not only allows us to pick the size, but we can pick the font as well. This is highly productive for those that need better viewing during development, but is mighty helpful to presenters doing demonstrations at a conference. You can also control the color of the fonts for non-default members, custom properties, and instance properties.

IntelliSense

IntelliSense was the big ticket item in Visual FoxPro 7 and is a huge productivity gain for developers. The biggest savings is the reduced need to hit the F1 key to bring up help. I use help probably 25% of the time that I use it in VFP 6.

ActiveX Controls

Understanding the object models of ActiveX controls is not difficult, but remembering all the properties, events, and methods can be a chore. So how can a developer get IntelliSense to do all the heavy lifting? The key is to declare a memory variable. Once it is declared, the object is live in the editor.

```
LOCAL loWord AS "Word.Application"

loWord = CREATEOBJECT("Word.Application")
loWord.Documents.Open(GETFILE("doc"))
loWord.ActiveDocument.Range.Bold = .T.
```

It should be noted that you do not have to declare the memory variables in the Command Window, only the editors. All ActiveX references a live in the Command Window since the `CREATEOBJECT()` is literally executed.

Custom expansion (VFP 8)

([Example: IntellisenseCustomExpansion.prg](#))

Custom expansion refers to the feature of IntelliSense that developers can auto expand a “word” in the same manner as Visual FoxPro commands are expanded as soon as Visual FoxPro recognizes the command and the developer hits the spacebar, or tab key. The custom ones are ones that Visual FoxPro developers define. This feature was available in VFP 7, but only had a few custom expansions (MC [Modify Command], MF [Modify File], and DC [Define Class]). Visual FoxPro ships with a list of new commands.

Table 2.. Here is a complete list of Custom Expansion commands that shipped with the marketing beta of VFP 8.

Custom Command	Expanded Code
DOCASE	DO CASE CASE OTHERWISE ENDCASE
DOWHILE	DO WHILE ENDDO
FOREACH	FOR EACH ENDFOR
FOREND	FOR ENDFOR
IFEND	IF ENDIF
IFELSE	IF ELSE ENDIF
SCANEND	SCAN ENDSCAN
TEXTEND	TEXT TO NOSHOW TEXTMERGE ENDTEXT
TRYEND	TRY CATCH FINALLY ENDTRY
WITHEND	WITH ENDWITH
WS	(expands out code to instantiate the selected webservice)

List Members, Quick Info

The List Members is the dropdown list of properties, events, and methods that are related to the current object hierarchy. It is displayed automatically when the object is specified, and can be manually initiated by the Ctrl+J shortcut. The Quick Info is the related help tooltip with the parameter list (including bolding the current parameter). It is displayed automatically as well, and can be manually initiated by the Ctrl+I shortcut. In VFP 7, when the backspace was keyed in we lost both the List Members and the Quick Info which meant that we needed the shortcut keys to get the IntelliSense started again.

The List Members does not stop functioning in VFP 8 when the backspace is keyed, it goes backwards in the list (incremental seaching) as you remove characters off the property, event, or method. This saves a ton of time not remembering the shortcut key to List Members.

C++ Operators

If you have developed in C++ you might be aware of the incrementor/decrementor operators. These are shortcuts to expand out common code like incrementing a numeric memory variable by typing in the memory variable with ++ added to the end of the variable. For instance:

```
lnCounter++
```

expands into:

```
lnCounter = lnCounter + 1
```

Table 3. The operators available as a default of IntelliSense

Operator	Feature	Example
++	Subtract 1 from the memory variable.	<code>lnCounter = lnCounter + 1</code>
--	Subtract 1 from the memory variable.	<code>lnCounter = lnCounter - 1</code>
+=	Add value to existing memory variable, leave value to be typed in.	<code>lnCounter = lnCounter +</code>
-=	Subtract value to existing memory variable, leave value to be typed in.	<code>lnCounter = lnCounter -</code>
*=	Multiply value to existing memory variable, leave value to be typed in.	<code>lnCounter = lnCounter *</code>
/=	Divide value to existing memory variable, leave value to be typed in.	<code>lnCounter = lnCounter /</code>

The feature is only activated with the spacebar, the enter key does not expand the operators. If you find this annoying you can turn it off via the Advanced page of the IntelliSense Manager.

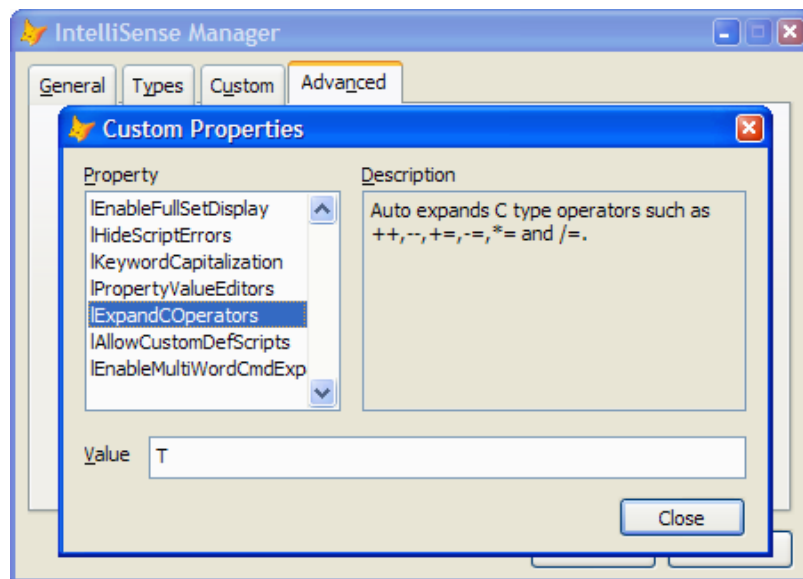


Figure 26. The C operator expansion is a configuration option available on the Custom Properties dialog accessed via the Advanced page.

Works inside of WITH...ENDWITH & FOR EACH (VFP 9)

The **WITH...ENDWITH** is a productive construct in Visual FoxPro because it reduces the amount of typing necessary in code. This was the case before IntelliSense came along and reduced the amount of typing we do to write our programs. Unfortunately, the two powers of productivity were mutually exclusive until Visual FoxPro 9. Now with the new **AS** clause on the **WITH** statement the object is “declared” and IntelliSense works.

The same happens with the **FOR EACH** command.

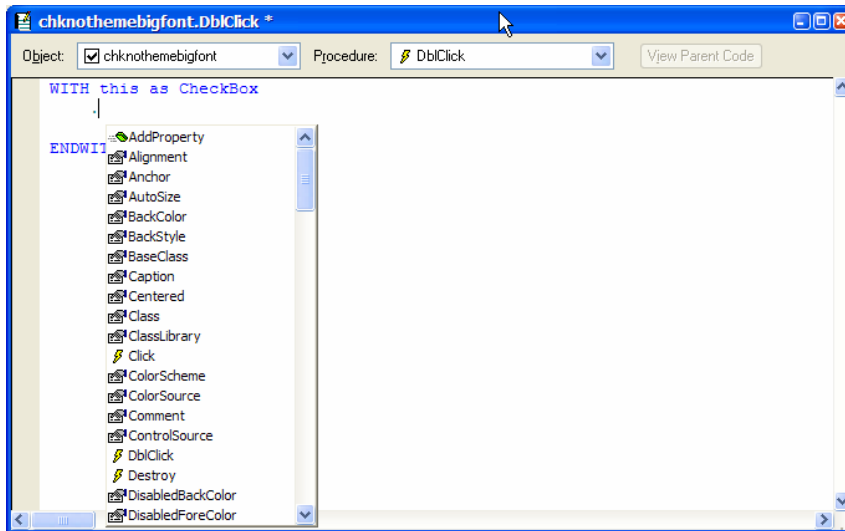


Figure 27. IntelliSense works inside of **WITH...ENDWITH** as long as you define the type of object you are working with in the editor. This is similar to the variable declaration process and interaction with IntelliSense.

_VFP.EditorOptions property persistent via FoxUser (VFP 9)

The EditorOptions define how your instance of Visual FoxPro responds with respect to IntelliSense. Previous to Europa, developers who wanted to deviate from the default ("LQKT") needed to set these programmatically. Now developers can set this once and have the setting persist via the FoxUser table. Naturally, this requires you to use a resource file if you want the settings restored the next time you start Visual FoxPro.

Editor option	cOptionString	Default	Runtime
List Members (Manual)	l	Off	No
List Members (Automatic)	L	On	No
Quick Info (Manual)	q	Off	No
Quick Info (Automatic)	Q	On	No
Enable Hyperlinks (CTRL + Click to follow the link)	K	On	Yes
Enable Hyperlinks (Click to follow the link)	k	Off	Yes
DragDrop Between Words	W	Off	No
Designer Value Tips	T	On	No

Designer Tips

Menu Designer

Support for moving menus (VFP 8)

The Menu Designer user interface has not been enhanced much over the years, but this is one feature that has been asked for often and a very welcome addition to our productivity. Previously developers had to cut and paste menu bars from one menu pad to another. Now you can select the menu item (pad or bar), press the Move Item button, and select the menu item that you want the selected item moved under (see **Figure 28**). The item is moved to the bottom of the list so you need to edit the section of the menu that you moved the item to so you can position it where it belongs. We also recommend checking the hotkey.

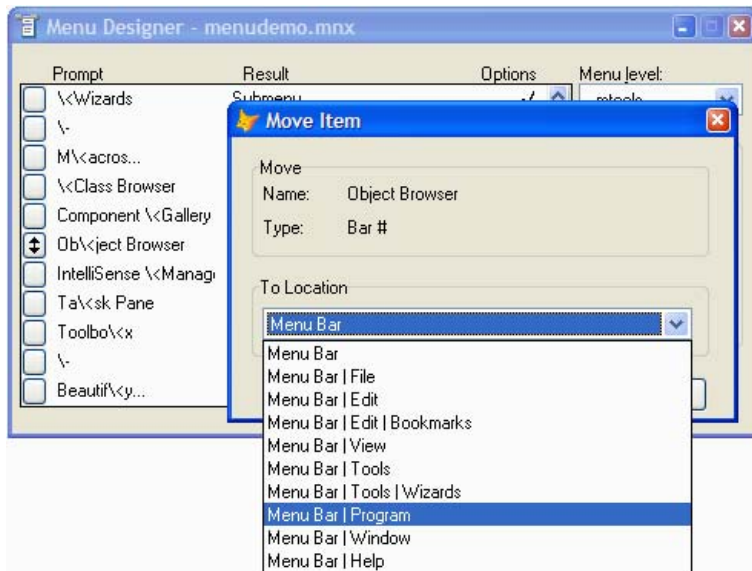


Figure 28. The new move functionality is a welcome feature in Visual FoxPro 8.

Support to hook in your own Menu Designer (VFP 9)

Creating a Menu Designer to replace the native functionality in Visual FoxPro is not something everyone will want tackle, but there are tool developer that will. Visual FoxPro developers have often requested changes to the Menu Designer and it was never prioritized. Now Microsoft has opened up a hook via the `_MENUDESIGNER` system memory variable so tool developers can natively run their designers when VFP developers interact with the Project Manager or `MODI MENU` in the Command Window.

Builders

The Builder technology has been available since Visual FoxPro 3. They are like wizards to set properties, but they are reentrant. They are an extension of the Property Sheet. This means that the builders read the existing properties of the object and allow you to make changes to the settings without resetting all the properties.

The first presentation I attended on builders was at the San Diego VFP conference in 1995. I was immediately impressed how the technology could impact my productivity. It is strange that we have seen very little implementation since the introduction. The builders that ship with Visual FoxPro have been considered weak.

All the source code to the builders is included in the VFP XSource directory. If you do not find the builders to your liking you can always change them or customize them to your needs.

So what builders are worthwhile and assist in productivity? Here is my list of builders that have increased my productivity:

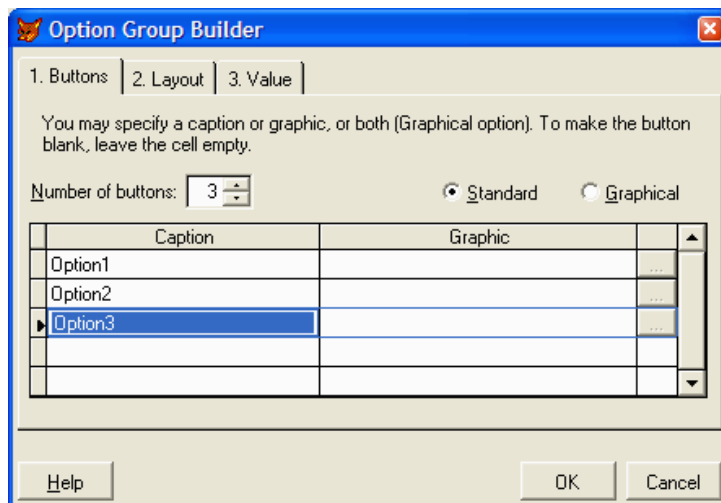
Referential Integrity (RI)

The Referential Integrity Builder is the fastest way to build the RI code for a Visual FoxPro database. It generates code that is not exactly optimized and can write code that will exceed the 64K compiled code barrier, but considering the alternative of writing your own code it is a good starting point.

I can say that one of the best developers in the FoxPro Community is Doug Hennig. He has written a replacement for the standard Visual FoxPro Referential Integrity Builder. His builder is based on the VFP RI Builder, but fixes a couple of bugs, and integrates Steve Sawyer's superior Referential Integrity code (called NewRI.prg). More details on the builder and the implementation are in a white paper from Stonefield that can be downloaded from <http://stonefield.com/pub/devtools.zip>.

Option Group

The Option Group object has a number of productivity killers built into the object. First, we never seem to have just two options so we have to add more options. Once we add them we need to traverse the Property Sheet to change the *Caption* properties, and then there is the tedious task of lining up all the options and make sure they are all evenly spaced.



The Option Group builder actually assists in quickly adding options, and changing the *Captions* on the Buttons page. The Layout page allows developers to determine if they want the options aligned vertically or horizontally, and how many pixels to evenly distribute them.

DataEnvironment (VFP 8)

The DataEnvironment builder productivity is in selecting the datasource. Using the interface to select the datasource, the builder makes the corresponding property settings and writes code in the *BeforeOpenTables()* method

```
*** Select code: DO NOT REMOVE
set multilocks on
***<DataSource>
This.DataSource = sqlstringconnect([dsn=BBCMKADS;])
***</DataSource>
*** End of Select code: DO NOT REMOVE
```

CursorAdapters (VFP 8)

The CursorAdapter class is a class that offers support for handling a wide range of local or remote data source as a native VFP cursor. Datasources supported include the following:

1. VFP tables
2. Open Database Connectivity (ODBC)
3. ActiveX Data Object (ADO)
4. Extensible Markup Language (XML)

The builder has nearly all the features that you find when creating a view, but without the overhead of a database container. This is one builder that is almost indispensable. Yes, you can create a CursorAdapter class and set all the corresponding properties, but it is far more productive to use the builder.

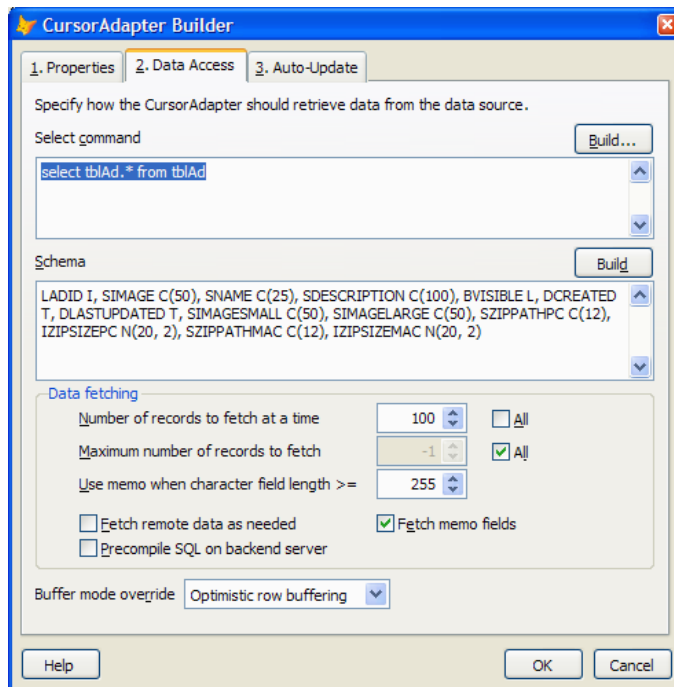


Figure 29. The CursorAdapter Builder can be called from a cursor in the dataenvironment or from the Class Designer when editing a Cursor Adapter.

Create your own builder

It is not in the scope of this session to explain builders and the implementation of builders, but they can assist productivity. For a detailed explanation of builders we recommend the developer tools article noted in the Referential Integrity builder section earlier in this white paper. Doug has done a great job explaining this complicated topic. Besides the normal way of registering a builder in the Builder.dbf, there three other ways of integrating a builder into Visual FoxPro: through a program, using the custom property BuilderX, and using the data driven BuilderD technology.

Write a program

[\(GenProjecthook.prg\)](#)

Most developers think of builders as a form with objects to interact and set properties. This does not have to be the case. The **ASELOBJ()** function gains the object references in the designer for a form interface, but it gains the object references for any selected objects in the foremost designer.

One example is to programmatically subclass an object and set properties.

Listing 2. This is a partial listing of the *GenProjectHook* program.

```
CREATE CLASS (tcProjectHook) OF ccPROJECTHOOKSLIB ;
    AS ccPROJECTHOOKSBASE FROM ccPROJECTHOOKSLIB NOWAIT

* Set the class property for the projecthook cFieldMappingCategory property
ASELOBJ(laProjectHookRef, 1)

IF TYPE("laProjectHookRef[1]") = "O"
    laProjectHookRef[1].cFieldMappingCategory = tcProjectFieldMappingConfig
ENDIF

* Make sure the reference to the projecthook is released
RELEASE laProjectHookRef

* Handle the VFP Windows that open with no Resource File
IF WEXIST("PROPERTIES")
    RELEASE WINDOW "Properties"
ENDIF

IF WEXIST("FORM CONTROLS")
    RELEASE WINDOW "FORM CONTROLS"
ENDIF

* Close the newly created class opened in Class Designer
* The keystrokes are "buffered" until all classes are created
KEYBOARD '{CTRL+W}'

* Added to close the class designers down
DOEVENTS()
```

BuilderX

(BuilderDemo2.scx, demo.vcx::frmCheckboxBuilder)

You can add a property called BuilderX to all classes. The native builder technology will use the setting in this property to run the builder designated. The property can have a program or a class library/class name combination separated by a comma. VFP will use this as the builder and ignore all the builders registered in the BUILDER.DBF table for that object.

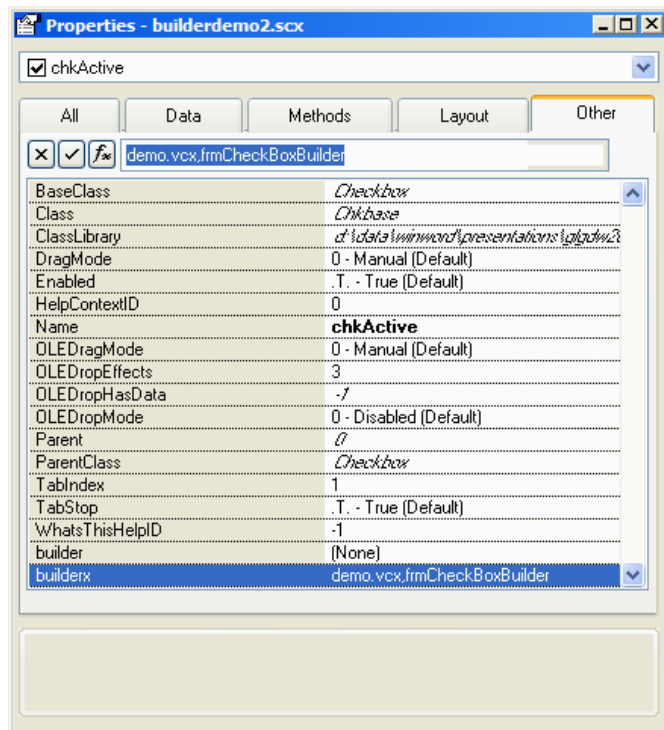


Figure 30. This example shows the class with the custom BuilderX property.

The base code in the builder form (frmCheckboxBuilder) is in the Load() event method:

```
ASELOBJ(thisform.aChange)

IF VARTYPE(this.aChange[1]) # "L"
    IF LOWER(this.aChange[1].BaseClass) = "checkbox"
        this.Caption= this.aChange[1].parent.Name + "." + ;
                        this.aChange[1].Name + " - " + this.Caption
    ELSE
        MESSAGEBOX("Object selected is not a checkbox.", 0 + 16, "Checkbox Builder")
        RETURN .F.
    ENDIF
ELSE
    MESSAGEBOX("No object selected for this builder.", 0 + 16, "Checkbox Builder")
    RETURN .F.
ENDIF

RETURN .T.
```

BuilderB/BuilderD

BuilderB is a technology developed by Ken Levy that is now incorporated into Visual FoxPro. It is a set of classes that allow developers to rapidly develop builders. Each builder starts with a form that is a subclass of the base BuilderB form. You add controls to the subclassed form for each property you want to maintain. This is faster than developing a new builder form from scratch each time, but can still be tedious and time consuming.

As fast as BuilderB made development of builders, it was replaced by a newer builder technology, the third generation of builder technology called BuilderD (for “Dynamic”, but I like to think of it as “Data”). This builder technology is data driven using some metadata stored in the BuilderD table (VFP\Wizards\BuilderD.dbf). I have added a couple of records to my BuilderD table for the session demonstrations:

Type	Id	Links	Classname	Classlib
CLASS	edtBase	BackStyle AllowTabs IntegralHeight BuilderX MouseIcon	edtBase	d:\data\winword\presentations\glgdw2002\getmoreproductivewithvfp\examples\demo.vcx
CLASS	cmdBase	Default Cancel	cmdBase	d:\data\winword\presentations\glgdw2002\getmoreproductivewithvfp\examples\demo.vcx

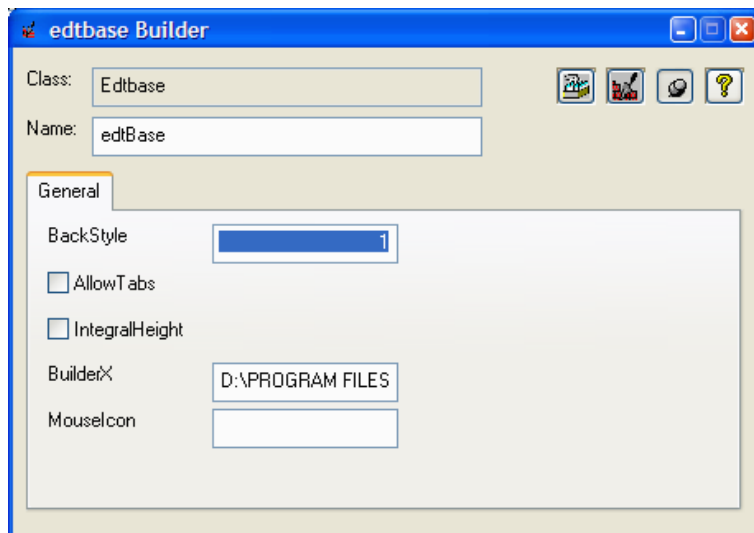


Figure 31. The results of the BuilderD metadata changes on the builder user interface for the edtBase class.

In literally minutes I can create a builder for any custom class I develop. Talk about enhancing productivity.

Tab Order (VFP 8)

When interactively setting the tab order for a form or class, Visual FoxPro now shows the most recent control selected. No more guessing what the last control was. Unfortunately there were not further improvements to the Tab Order process, like starting in the middle and having the controls correct the order from the restart of the tab order settings.

Tab Order on properties for form/class designer (VFP 9)

There have been two different ways to change the tab order of controls in the Form and Class Designers since VFP 3.0. Some developers prefer the By List, some developers prefer the Interactive method, and some of you prefer to use one under certain circumstances and the other under different circumstances. If you like to change frequently you might find toggling between the two styles a tedious task. Prior to Visual FoxPro there were two ways to toggle your tab order method, either use the Options dialog, or write some code to toggle the Windows Registry entry.

Visual FoxPro 9 extends the Tab Order menu option found on the View pad (Figure *?). Your selection in the Visual FoxPro Options dialog is now meaningless, and you have to select which one you want to use each time you set the tab order.

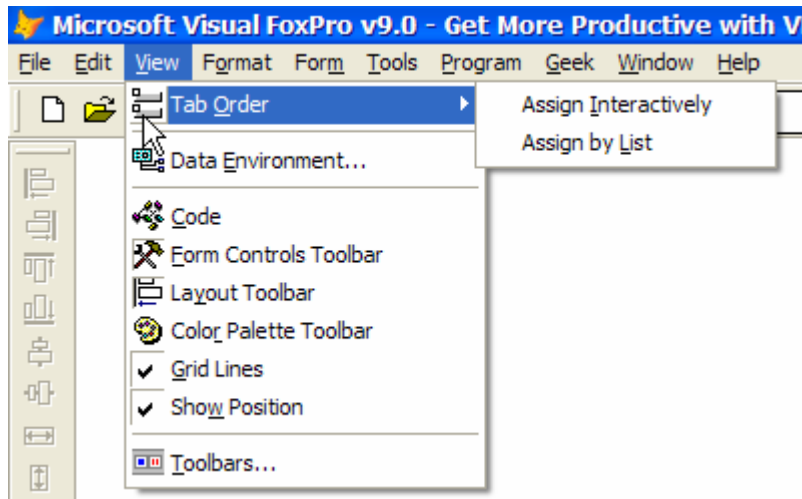


Figure 32. The Form and Class Designer now allows you to select which style of Tab Ordering you want from the View Menu.

VFP Design Surface (VFP 8)

No maximum limit for design area

Previous to Visual FoxPro 8, developers had to configure the maximum design area of the Form and Class Designer. Many developers selected the maximum size their customer forms could be. You could select from the popular desktop resolutions. The problem presented is that you might have a monitor that could support resolutions greater than the 1600x1280, but that is the largest designer surface VFP supported. What can you do with these requirements? Nothing.

VFP 8 removes this limitation all together.

Debugger

The Visual FoxPro developer typically spends a significant amount of time debugging code. Optimizing productivity with the debugger can save a lot of time.

Set the debugger configuration to factory settings

(Example: [ClearDebuggerSettings.prg](#))

There are many settings and customization capabilities for the debugger in Visual FoxPro for developers to adjust. Every once and a while you might want to just get back to the basics. This can be accomplished with one command:

```
CLEAR DEBUG
```

This command clears all breakpoints, restores the Debugger windows (Trace, Locals, Call Stack, Watch and Output) to their default positions, clears the expressions in the Watch window, and clears the Output Window. This works in the debugger frame or the FoxPro frame.

There is one reason why you might want to get back to the factory settings. Occasionally we run into C5 errors when using the debugger. Many of the causes have been tracked down and fixed over the years by the Fox Team, but others still linger. A common resolution is to turn off the FoxPro resource file and see if it clears the C5 problem. If the error goes away it is concluded to be a problem with one or more of the debugger preferences or settings stored in the FoxUser table. One solution typically presented is to delete the FoxUser files and start clean. This is a bit drastic since there are specific records in the FoxUser file that can be removed that provides the same effect. Here is the code that can be run to fix the problem.

```

LOCAL lcOldResource
lcOldResource = SYS(2005)

SET RESOURCE OFF
USE (lcOldResource) EXCLUSIVE ALIAS curResource

DELETE ALL FOR id = "BPOINTS"
DELETE ALL FOR id = "DBGFRAME"
DELETE ALL FOR id = "DEBUGFRAME"
DELETE ALL FOR id = "DEBUGGER"
DELETE ALL FOR id = "ETRACK"
DELETE ALL FOR id = "F_DBGWINDOW"
DELETE ALL FOR id = "WATCHEXPR"

PACK

USE IN (SELECT("curResource"))

SET RESOURCE ON
SET RESOURCE TO (lcOldResource)

```

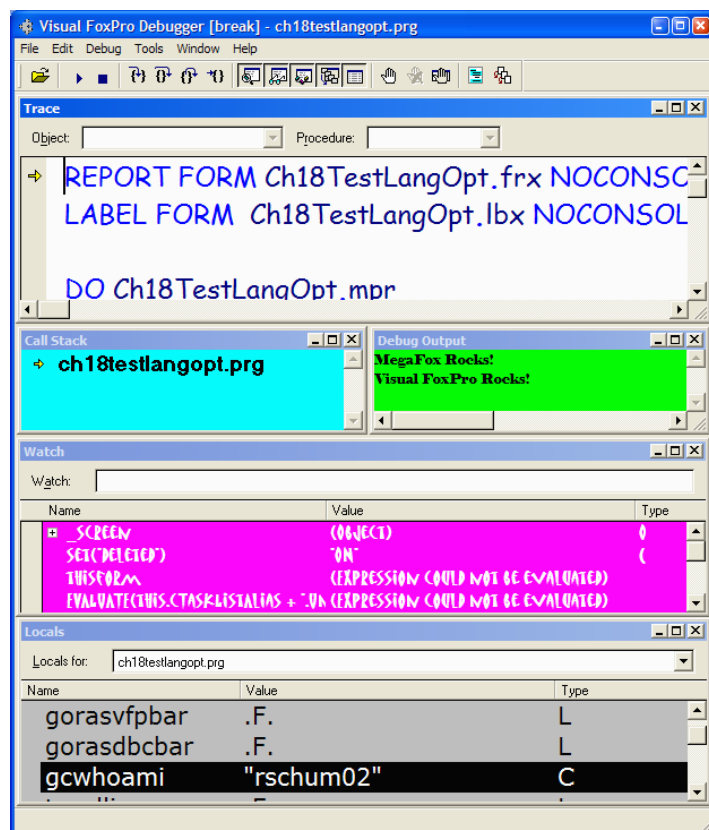


Figure 33. The debugger can be configured to select fonts and colors (foreground and background).

You will find that the color and font settings are not reset after deleting records in the resource file. The reason for this is that the settings are saved in the Windows' registry under the following key:

HKEY_CURRENT_USER\Software\Microsoft\VisualFoxPro\7.0\Options

Table 4. Registry values to store the debugger window colors.

CallstackChangedColor

TraceBreakpointColor

CallstackFontName	TraceCallstackColor
CallstackFontSize	TraceChangedColor
CallstackFontStyle	TraceExecutingColor
CallstackNormalColor	TraceFontName
CallstackSelectedColor	TraceFontSize
LocalsFontName	TraceFontStyle
LocalsFontSize	TraceNormalColor
LocalsFontStyle	TraceSelectedColor
LocalsNormalColor	WatchChangedColor
LocalsSelectedColor	WatchFontName
OutputFontName	WatchFontSize
OutputFontSize	WatchFontStyle
OutputFontStyle	WatchNormalColor
OutputNormalColor	WatchSelectedColor
OutputSelectedColor	

The color settings are not exactly straight-forward since they are comma-delimited lists of non-standard **RGB ()** sequences (first three are the foreground color and last three are the background color), followed by the determination of the foreground and background colors and whether they are automatic or not automatic (specified by the RGB selection). The font attributes (name, size, style) can be easily handled programmatically via a registry class like the one shipped with Visual FoxPro as part of the Fox Foundation Classes. The font style is set to zero for normal, one for bold, two for italic, and three for bold and italic.

Save and restore the configuration of the debugger

(Example: [DebuggerConfigSaved.dbg](#), [VFPStandard.dbg](#))

A Visual FoxPro developer can go through a lot of work configuring the debugger with the settings for the watch window, developing the exact breakpoints needed for an application or module, and selecting certain events to be tracked. The settings can change depending on the application or a specific module in an application. We can delete expressions from the watch window and enter in new ones as we test various modules, we can toggle breakpoints in use and not in use, and we can move events that are tracked on and off the list. Another way is to save the exact configuration for the module and later load the configuration without the need to reenter the expressions or toggle the breakpoints.

This is accomplished via the Debug frame only. Using the menu, you can select the File | Save Configuration... to create a file. The file save dialog will default to the current Visual FoxPro directory. To restore a previous configuration you use the File | Load Configuration... menu option. The file contents are stored in an ASCII text file. Here is an example:

```
DBGCFGVERSION=4
WATCH=_screen
WATCH=set("deleted")
WATCH=set("path")
WATCH=thisform
WATCH=curdir()
WATCH=recno()
WATCH=eof()
WATCH=_vfp.ActiveProject
BPMESSAGE=OFF
BREAKPOINT BEGIN
TYPE=2
CLASS=
LINE=0
EXPR=EOF("curReport")
DISABLED=0
EXACT=0
BREAKPOINT END

BREAKPOINT BEGIN
TYPE=3
CLASS=
```

```
LINE=0
EXPR="MAIN"$PROGRAM( )
DISABLED=1
EXACT=0
BREAKPOINT END

EVENTWINDOW=ON
EVENTFILE=
EVENTLIST BEGIN
Activate, Deactivate
EVENTLIST END
```

You can manipulate the contents safely and reload the configuration. Make backups of this file if you are worried of breaking the layout.

Change values of memory variables in the debugger

Have you ever been debugging some code to find out that a value of a memory variable is not as expected and wished to see how the rest of the code would work if the value was corrected at that point? You can literally change the value of a memory variable as you are stepping through the code in the debugger without the Command Window.

This can be done in the Locals window or the Watch window. Select the memory variable in the Locals window by clicking the mouse on the entry. Click a second time in the Value column which activates the edit mode for the value. Enter in the value you want the memory variable to be and move off the entry via the tab key, enter key, or via a mouse click on another item in the debugger.

Now stepping through the code will use the new value and you will be able to evaluate how well the code is working when expected values are used.

Drag and Drop

There is a number of drag and drop opportunities we find ourselves using that we have not seen documented.

You can highlight code in the Trace window and drag it to the Watch window. You can also do the same for variables in the Locals window. The expressions can either be dropped in the textbox or in the evaluated list. If you drop it in the list it is automatically added to the list. If you drop it in the textbox you need to press the enter key to add it to the list. Naturally you will want to drop expressions that can be evaluated by the Watch window.

You can drag and drop expressions from the Watch window list to the Watch window textbox. This can be useful when you want to add another expression for a different property on the same object or want to add additional levels of containership to the expression.

The contents of the Trace, Watch, Locals, and Debug Output windows can be dragged to the Command Window. This can be handy when testing interactively and you want to jump into the Command Window to do further evaluations of the running code.

Changing Expressions

If you have a syntax error or a typo error in the Watch window you can click twice to edit the expression directly in the window. This can be helpful when you entered in a long containership hierarchy and need to correct it quickly, without the need to enter in another entry in the Watch window.

Quick access to the property values of a specific object

We know this is an old tip from as far back as 1995, but we have had a few new developers cross our paths since then and see some value in repeating the tip once more. The `sys(1270)` function gets an object reference to the object directly beneath the mouse pointer. We set up a couple of hotkeys to get the object reference:

```
ON KEY LABEL F8 ox = SYS(1270)
ON KEY LABEL Ctrl+F8 RELEASE ox
```

Now you have a reference to look at in the Locals window, **DEBUGOUT**, display in a **WAIT WINDOW**, or even print to the Visual FoxPro desktop. In the debugger you can drill down to look at specific public properties. From the Command Window you can display the property settings as well as call the object's public methods. Make sure to release the object, otherwise the object will not be able to be destroyed.

IntelliSense in the Watch window (VFP 8)

IntelliSense is so handy and in my experiences eases the use of the editors to the point that I use the help file approximately 70% less than I did with VFP 6. The frustrating part is when I am using the debugger and want to examine a particular object property. You can get a reference to the object or drill down through a reference to an object reference in the Local window if one exists. Another approach was to add in a reference to the object in the Watch window, but in VFP 7 you needed to remember the entire object hierarchy as well as the property name you wanted to examine. In VFP 8 the Fox Team added the power of IntelliSense to the Watch window. Saves me time each time I use the Watch window.

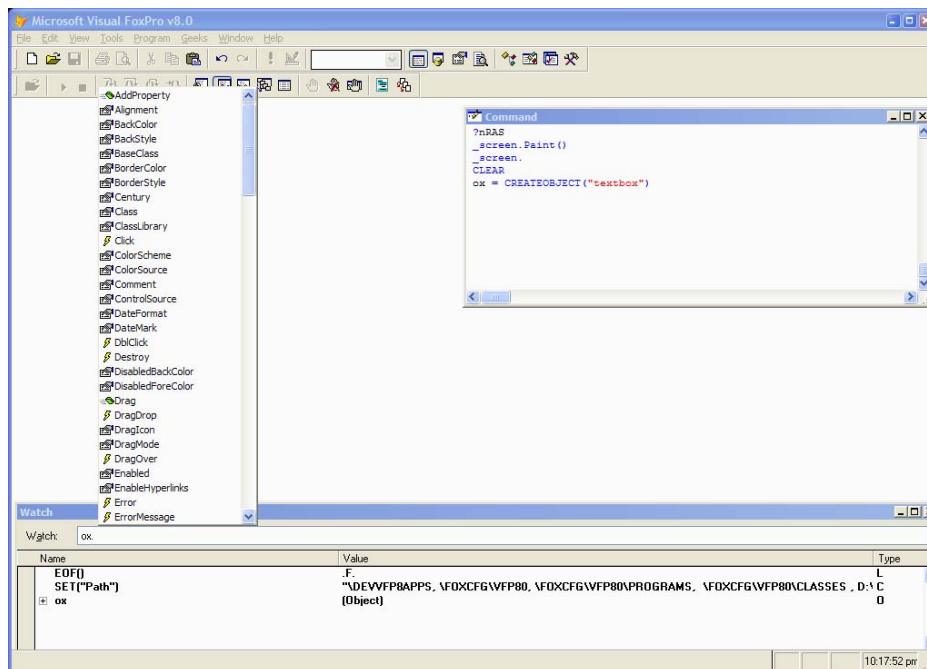


Figure 34. The Watch window has the same IntelliSense capabilities as the different editors.

Coverage Profiler

The Coverage Profiler shows two views of executed code for analysis. The first view is to show the “coverage”: what code was executed during the test and more importantly, what code was not executed. The profiler mode shows performance bottlenecks. Using this tool allows developer to quickly find performance issues. I figure that I save hours a year using this to determine where my slow code is, rather than guessing, plugging in a bunch of **WAIT WINDOWS** or **DEBUGOUT** commands, or stepping through the code.

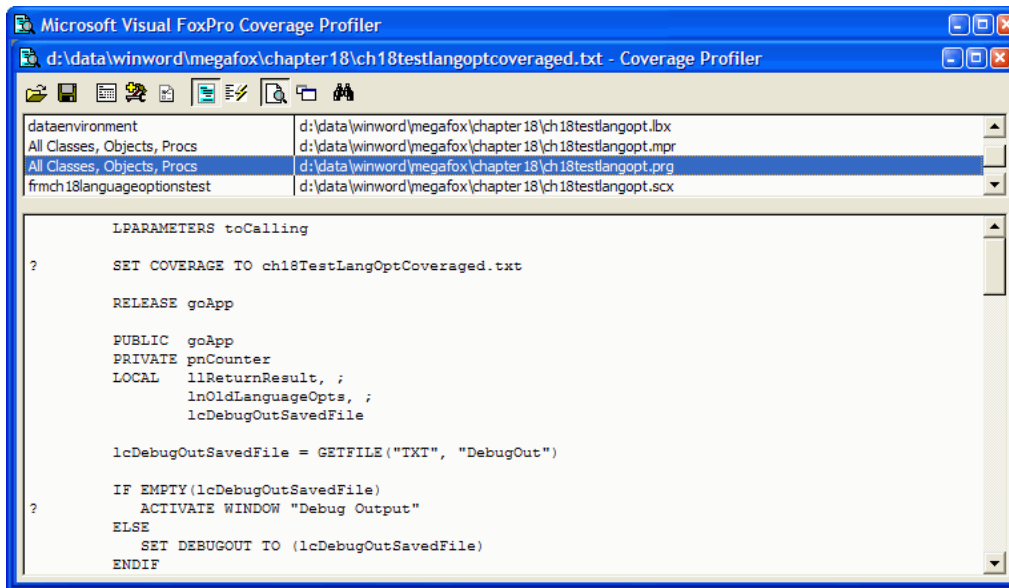


Figure 35. The coverage mode shows which lines of code were executed, and more important, which lines of code were not executed during the test.

The coverage mode is more useful in testing because it helps us ensure that we execute the correct code as we step through our test plan. One tip we like to recommend is changing the character that represents the code that was not executed. We use the question mark (?) (see **Figure 35**) because it reminds us to ask the question: Why was this code not executed during this test? You can change this in the Coverage Profiler Options by pressing the fifth button from the left on the toolbar at the top of the main form.

VFP IDE

The Visual FoxPro Interactive Development Environment is the most extensible development environments around.

BROWSE

BROWSE NOCAPTIONS will display the browse window with the column names in the headers. Why is this a productivity tip? It saves me from having to do a **LIST STRUCTURE** to see the column name when I cannot remember what captions are assigned to what columns.

BROWSE memo tips (VFP 9)

You know that the **BROWSE** command is very powerful when editing data in the IDE, but you might be frustrated by the word “Memo” telling you there is data in the column. You need to open it up and view it. In VFP 9 Microsoft added memo tips which are ToolTipText displayed for memo columns. This feature is not available in a grid, but you can expose the data by adding a control to the column so it is not needed.

Procname	Proclineno	Lineno	Colpos	Matchlen	Abstract	Recordid	Updfield	Checked	Noreplace	Timestam
memo	0	0	0	0	Memo			F	F	10/17/200
memo	0	0	0	0	Memo			F	F	10/17/200
memo	0	0	0	0	Memo			F	F	10/19/200
memo	0	0	0	0	Memo			F	F	10/17/200
memo	28	28	59	5	Memo	RQE0MEQNV	PROPERTYNAME	F	T	10/26/200
memo	0	0	0	0	Memo			F	F	11/18/200
memo	0	0	0	0	Memo			F	F	10/17/200
memo	0	0	0	0	Memo			F	F	10/17/200
Memo	6	29	4	3	Memo	OLQ1BP249	METHODS	F	F	06/15/200
memo	0	0	0	0	Memo			F	F	11/08/200
memo	1	1	4	3	Memo	05D18V80X	OBJNAME	F	T	10/26/200
Memo	87	32	10	3	Memo	0F015TJ4X	METHODS	F	F	11/18/200
Memo	29	208	15	3	Memo	0F015TJ4X	METHODS	F	F	11/18/200
Memo	31	210	15	3	Memo	0F015TJ4X	METHODS	F	F	11/18/200
memo	0	0	0	0	memo			F	F	10/19/200
memo	0	0	0	0	memo	07/22/2001 RAS 1.0 Fixed bug with truncation of project path		F	F	10/19/200
memo	0	0	0	0	memo			F	F	10/19/200
memo	0	0	0	0	memo			F	F	11/09/200
Memo	1	1	2	3	Memo	0S1C1V5Q	PROPERTYNAME	F	T	11/09/200
memo	0	0	0	0	memo			F	F	11/09/200
memo	85	85	30	3	Memo			F	F	07/12/200
memo	95	95	62	3	Memo			F	F	07/12/200
memo	96	96	20	3	Memo			F	F	07/12/200
memo	87	87	30	3	Memo			F	F	07/29/200
memo	0	0	0	0	Memo			F	F	10/21/200
memo	0	0	0	0	memo			F	F	10/17/200
memo	0	0	0	0	memo			F	F	06/15/200

Figure 36. The memo tips save time by eliminating the need to open the memo window and navigating to the record. All you have to do is move the mouse pointer over the record's memo column to see the content.

Macros / ON KEY LABEL

The Visual FoxPro macro recorder has been available since the earliest days of FoxPro, all the way back to the DOS version. If you have repetitive keystrokes we recommend that you look at the Tools | Macros option. Macros are available to production applications, not just the development environment.



Figure 37. The Visual Macro recorder allows you to insert literals or pauses when you prompt the recording to stop.

Once the macro is recorded you can edit it via the Tools|Macros... dialog.

The other option is to program keys using **ON KEY LABEL**. One of our favorite setups is to hook The Hacker's Guide help file as an alternative help file. Now we are one key away from the most useful help file around.

```
ON KEY LABEL ALT+F1 RUN /N c:\windows\hh.exe "D:\Hentzenwerke Books\hackfox7.chm"
```

Another set of keys that I like to program is F8/Ctrl+F8 to gain an object reference to the object just under the mouse.

```
ON KEY LABEL F8 ox = SYS(1270)
ON KEY LABEL CTRL+F8 RELEASE ox
```

This same productivity tip is discussed in the context of the debugger to get a reference to an instantiated object, but it can also be used to get an active object reference to an object in a designer (Form or Class). Each of the objects in the designer is a live object. Once you have the reference you can change the properties of the object via the Command Window. The full power of IntelliSense is also available through the object reference.

Dockable forms for custom tools (VFP 9)

Prior to VFP 9, the only forms that could be docked were the native VFP windows like the DataSession, Command, Property Sheet, and Document View. Now you can create forms that dock. You can dock to the sides of the VFP frame and in combination with other windows (both VFP windows and your own custom windows). This allows you to better organize your IDE which will make you more productive.

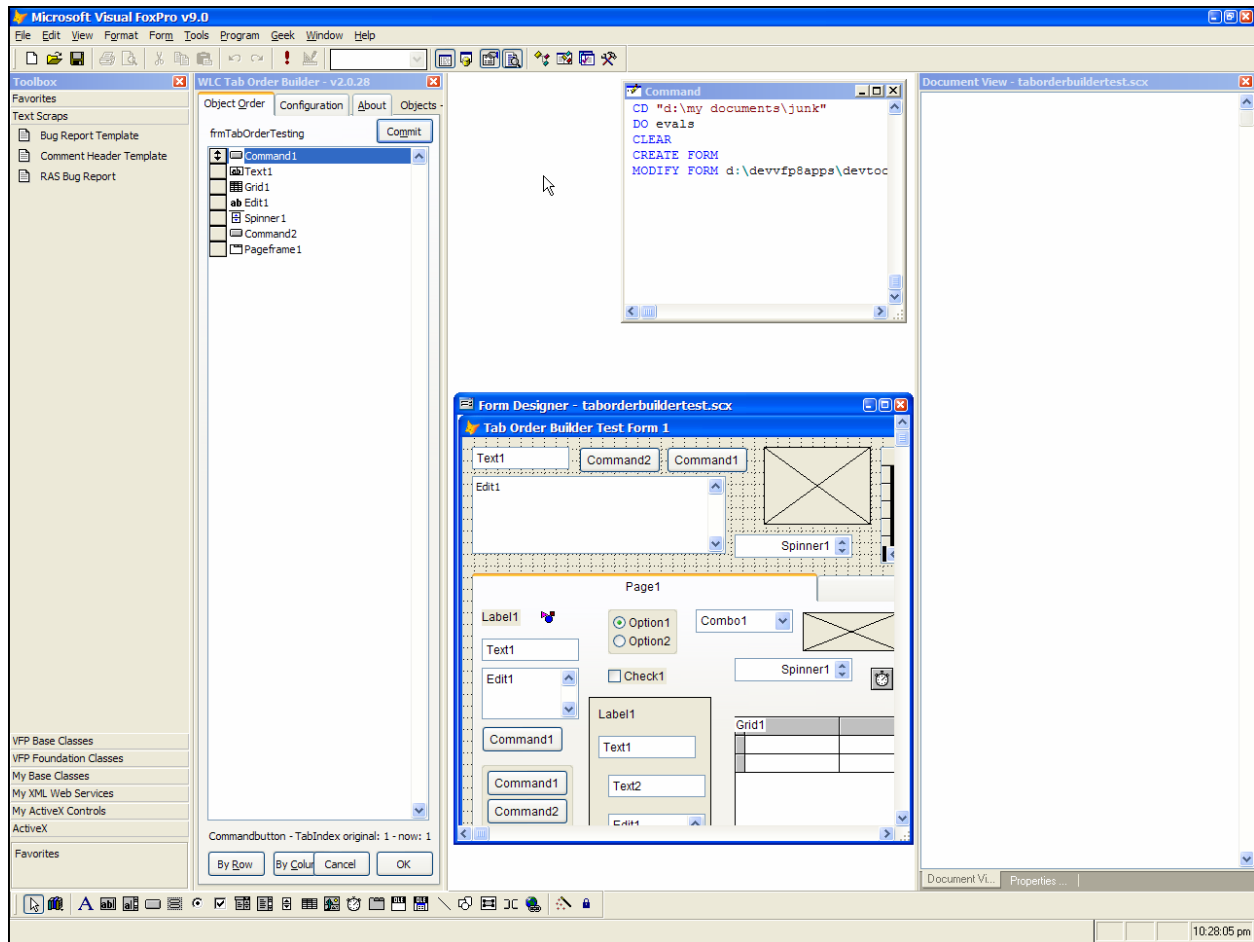


Figure 38. This figure shows the VFP ToolBox (based on a VFP form), the White Light Computing TabOrder Builder both docked on the left side of the IDE. The Document View and Property Sheet are docked on the right side together in one window.

SHIFT key and menus

Shift+File provides a Close All option to close all open forms

Shift + Format provides a mechanism to change the _SCREEN font instead of the normal editor font change.

C5 Errors slowing you down?

There is nothing worse than Visual FoxPro crashing on you while stepping through code during a debugging session for 30 minutes or editing the same program for an hour and losing your changes. While there is no way to guarantee that you will not run into Microsoft's "catch all" error handler "otherwise" code, we have a couple of things we like to try to eliminate them.

The first is a best practice concept: if you create an object, you need to release the object. This concept is often referred to as garbage collection. Visual FoxPro has gotten better at doing this over the years, but I can say that I saw much of Visual FoxPro's instability go away after adopting this practice. I have been doing this since early days of Visual FoxPro 5.

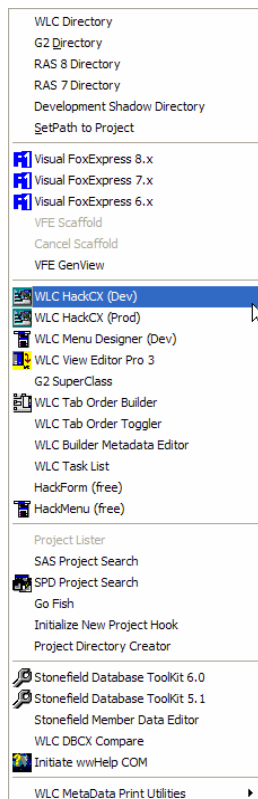
```
* Object Init()  
this.oParameter = toParameter
```

```
*Object Destroy()  
this.oParameter = .NULL.
```

The second way we find often eliminates C5 errors is to eliminate a possible corrupt FoxUser file. This can be tested by doing a **SET RESOURCE OFF**. If you notice that the repeating C5 errors disappear, then chances are that the resource file was corrupt. The unfortunate thing is that your settings saved in the resource file are lost, but it is better to stop the fatal crashes.

Use a developer menu to run repetitive tasks

Developers who have used Visual FoxPro for awhile recognize that you can write menus that are a single pad with custom options. I have used this technique to create a developer menu with all the different tools and commands that I was constantly entering into the Command Window. This saves probably a week of time a year avoiding the CD command to switch to a tool directory and then executing the tool. Even with the Most Recently Used lists that are part of IntelliSense I still save easily 30 minutes a week by creating my own shortcuts. This feature might be less useful for developers taking advantage of the new Task Pane in VFP 8.



When VFP 3 shipped I started with a toolbar with command buttons for each of my tools. The reason I implemented a menu instead is that a `CLEAR ALL` removes the toolbar from memory (see next section “Mopping up after a program crash”)

Mopping up after a program crash

This is one of those productivity tips that many Visual FoxPro developers learn early, but it is worth noting in the whitepaper for developers that do not have it in their toolbox. Often when our programs crash it leaves the development environment in a state of flux. Databases, tables, other and cursors can be left open, object references hanging, programs cached in memory,

```
CLEAR ALL
CLOSE ALL
CLEAR PROGRAM
```

`CLEAR ALL` releases all the current memory variables, removes all objects from memory, which, in turn, closes all private data sessions (closing all open cursors). `CLOSE ALL` closes all databases, tables, and cursors in datasession one (the default Visual FoxPro data session). `CLEAR PROGRAM` clears the compiled program buffer of the most recently executed programs and insures that Visual FoxPro will read the programs from disk the next time they are executed, rather than from the program buffer (will save you plenty of aggravation when changes you make in code are not reflected in a running program).

Cleaning up during transactions: if transactions are in progress, use the `END TRANSACTION` command for each level of transaction before issuing the `CLEAR ALL`, `CLOSE ALL`, and `CLEAR PROGRAM`.

Cleaning up during buffered updates: If buffered updates are in progress, use either the `TABLEUPDATE()` function or `TABLEREVERT()` function for each cursor with buffered updates before issuing the `CLEAR ALL`, `CLOSE ALL`, and `CLEAR PROGRAM`.

Developer Tools

G2 SuperCls

<http://geeksandgurus.com/>

<http://rickschummer.com/vfptools.htm>

Most Visual FoxPro developers are familiar with Ken Levy’s SuperClass utility. This utility works as a visible toolbar whenever the method editor is opened. The first toolbar button (see **Figure 39**) allows you to edit the code in the superclass (parentclass). You cannot open the superclass to edit code when the subclass is already open in the Class Designer. This tool provided developers with a way to peak at the code to see what they might be completely overriding or to determine how to extend behavior with a call to the superclass via a `DODEFAULT()` or a call up the class hierarchy via a direct call via the Scope Resolution Operator (`::`). Right-clicking on the Edit Superclass button will show all the classes in the hierarchy so you can select which level of code to review.

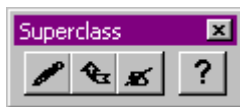


Figure 39. Original Superclass Utility by Ken Levy.

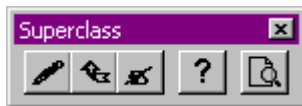


Figure 40. Upgraded Superclass Utility from Geeks and Gurus.

Drew Speedie, architect of Visual MaxFrame Professional extended this tool with an additional button. The functionality he added displayed all the code in each of the superclasses in one edit window. This code cannot be edited in any fashion, but it quickly allowed the developer to see all the code that would be executed provided that the appropriate calls were made in the hierarchy. However, this tool is shipped with VMP, and requires code in VMP to run. Since my partner Steve Sawyer and I are fond of Drew’s additional feature, but we are using another

framework at the moment, we decided to hack Ken's source code and plug in Drew's functionality in a generic manner, one that is independent of the VMP framework code.

```

» » » » IF .loList.uRetVal
» » » » llReturn=.T.
» » » » luUpdateValue=.GetLookupUpdateValue()
» » » » IF .NOT .ISNULL(luUpdateValue)
» » » » » IF .NOT .Value==luUpdateValue
» » » » » IF TYPE("oCursor.Alias")=.T_CHARACTER.AND .USED(oCursor.Alias)
» » » » » » SELECT(oCursor.Alias)
» » » » » » ENDF> » » » » »
» » » » » .Value=luUpdateValue
» » » » » ENDF
» » » » » ENDF
» » » » ENDF
» » » ENDF
» » ENDF
ENDWITH
RETURN llReturn

*****j:\ggproject\framework\vfe\vfeframe\libs\cdata.cfield*****
*****No code in this class*****
*****j:\ggproject\framework\vfe\ilibs\idata.ifield*****
*****No code in this class*****

```

Figure 41. Output of the code in the class hierarchy is displayed in a program editor for colorization.

WLC ProjectBuilder

<http://rickschummer.com>

The WLC Project Builder provides an interface that consolidates some of the Project options from the VFP Project Information dialog, some of the VFP Project Builder and Version dialogs, as well as some handy hooks to the WLC ProjectHook (if instantiated), and resolves some issues with **SET STRICTDATE** for production builds.

This tool was designed to handle some of my ideals when building development versions and building the "gold" code for production releases.

Feature	Development	Production
Recompile All Files	Default OFF	Default ON
Display Errors	Default OFF	Default ON
Debug Code	Default ON	Default OFF
SET STRICTDATE TO	2	1
Clean Printer Code from reports (FRXs) (note this is only available with RAS ProjectHooks)	Default OFF	Default ON

All the settings that are retained in the project file (Auto Increment, Encrypted Executable, and Version page) are read from the settings in the project. The Process Project Audit Trail, Clean Printer Information from Reports, Project Activate (only for VFP 6), and Field Mapping features do require the WLC ProjectHook activated for the project (included in the same file, just not required).

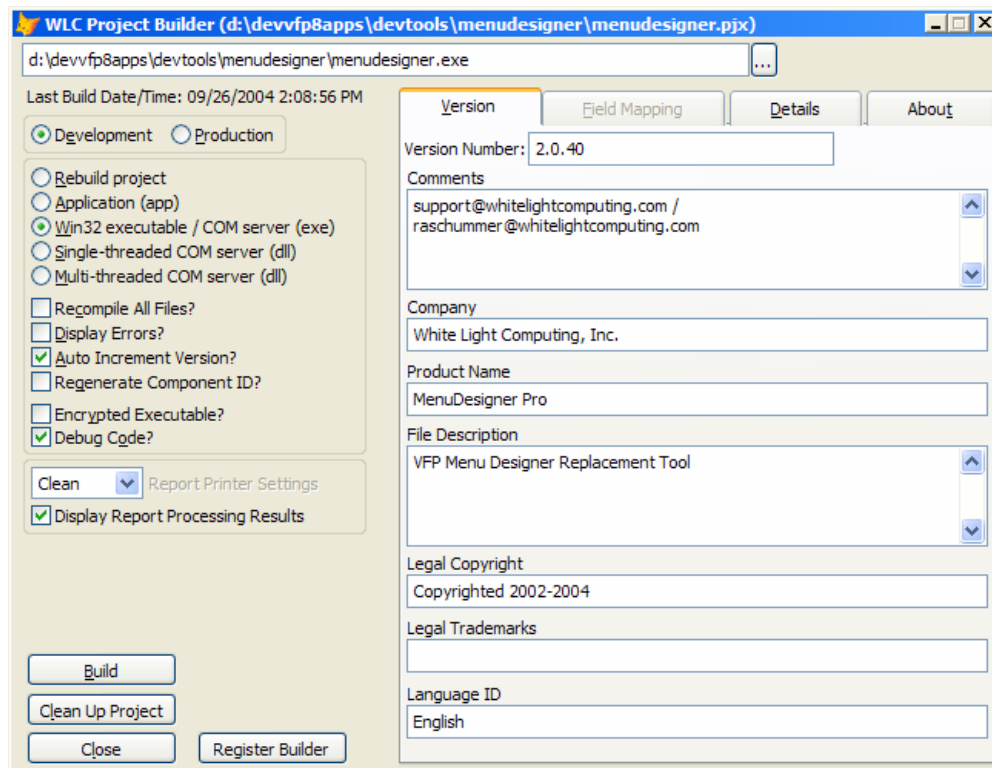


Figure 42. The RAS Project Builder saves time toggling between the various VFP Build Dialogs.

WLC Tab Order Builder

Have you ever been annoyed by the native Visual FoxPro Tab Order dialog? This is the dialog that you can use to move objects up and down the list to change the tab order within the Form or Class Designer. If you have long object names they get truncated and when you move an object up or down the list the icons for the objects you pass over are changed to the object you are moving. Is this a bug in VFP? Absolutely, and it has been there for many versions.

Now some developers prefer the interactive style of setting tab orders and have not used the "By List" style for years either because they find it annoying or they prefer the interactive mode. I prefer using both, each under different circumstances. The interactive version is great when you need to adjust most or all of the tab orders after initially developing the form or class. I prefer the list when I need to adjust a couple of items. If only one or two items out of 20 or more need to be changed it can be time consuming to change them all. Up to and including VFP 7, you could not tell which object you are currently on and if distracted during the setting, you likely needed to start over (VFP 8 now highlights which object you last set).

So I created the WLC Tab Order Builder to eliminate the need to use the "By List" tab order dialog in VFP. It does everything the native dialog does and more. Here are a list of advantages:

- Resizeable - you can see those long names
- Indicates the current tab order as well as the original order before you moved the object.
- Displays the object name of the container currently getting the tab order set (this is displayed just above the listbox).
- No icon changing when you move objects up and down the list.
- Shows a count of the objects in the list
- Restores the form size and position where you left it the last time you used it. You can turn this feature off if you like on the configuration page.

- Configuration page allows you to control whether errors are logged and where they are logged, if themes are used (VFP 8 and higher), and whether the configuration is saved on exit.
- The about page has hyperlinks to the various Web sites that this tool is available on and has a commandbutton to view any logged errors.
- Runs as a VFP Builder (using Builder technology) or as a standalone program from a menu (see details below).

This tool is a VFP Builder (ALL and PAGE builder). To register this tool as a builder run the executable and pass it .T. as a parameter. Unfortunately the PAGE builder is not working in VFP (any version, including VFP 8 SP1), but the ALL builder kicks in.

```
CD < folder where the tool is located >
DO G2TabOrderBuilder.exe WITH .T.
```

WORD OF CAUTION: VFP has a quirk in it if you only have one ALL builder. What will happen is that the only ALL builder will be run for any object that does not have any other builders registered. For instance, you do not have a SPINNER builder registered, the ALL builder will automatically run when you right-click on a spinner object and select Builder... This might not be what you want. I always recommend having two or more ALL builders so you can select which one runs or cancel if you really do not want a ALL builder running for the object you selected.

If you want to run the WLC Tab Order Builder from a menu (ideal for Visual FoxExpress developers since each VFE object has a specific builder and VFP builders are not currently accessible in the VFE development environment), you can add it to a custom developer menu or one of the VFP menu pads:

```
DEFINE BAR 19 OF geeks PROMPT "G2 Tab Order Builder" ;
    SKIP FOR !FILE("J:\GGProject\Tools\G2TabOrderBuilder.exe") ;
    MESSAGE "J:\GGProject\Tools\G2TabOrderBuilder.exe"

ON SELECTION BAR 19 OF geeks DO J:\GGProject\Tools\G2TabOrderBuilder.exe
```

Naturally you will need to replace the directories I have established.

This is ideal if your objects have a *Builder* or *BuilderX* property set. The builder defined in these properties always takes precedence to the builders registered in the Builder.dbf table. This is the situation that Visual FoxExpress developers face or objects included in your applications from the Fox Foundation Classes (FFC).

The builder will run for the currently selected object. If the object is a container it will present all the objects in the container. If it is not an object that contains other objects, then the builder will present the objects in the container that the selected object is in.

There is two quirks/issues with this tool. The first is with page objects. If you have a page in a pageframe, right-click on the page and select builder you will get a message "This builder cannot find a selected control with an appropriate base class." This message is getting displayed from Builder.app (the native Builder Manager application that controls the execution of builders). It basically fails with page objects. I have reported this to the VFP team as a bug and they are looking into it.

The second issue that exists is that OLEControls show up in the VFP Tab Order dialog and not in the WLC Tab Order Builder. The way this tool works is that it looks for the *TabIndex* property on the object. If the object does not have a *TabIndex* property it is skipped since it has no tab order associated with the object. The OLEControl does not have this property. I am not sure how the VFP team is handling this internally, but I am going to pursue it with them when I get a chance.

Conclusion

Productivity is something you all strive for when developing day-to-day in any developer tool. I hope I have succeeded in demonstrating a few tips and tricks in getting more productive in Visual FoxPro.

Special Thanks

I want to thank the guinea pigs that put up with the rehearsals to insure that this presentation was refined for primetime at the Great Lakes Great Database Workshop 2002 and DevEssentials 2004. The [Detroit Area Fox User Group](#) and [Grand Rapids Area Fox User Group](#) members provided excellent feedback to me and I really appreciate the frank and honest evaluations that were provided.

I want to thank Pamela Thalacker and Steve Sawyer who reviewed these notes before publication. Steve Dingle, tech editor of *MegaFox: 1002 Things You Wanted To Know About Extending Visual FoxPro* reviewed all the material in Chapter 18: Testing and Debugging, which some of this material was borrowed.

Copyright

Copyright © 2002-2004 Richard A. Schummer. All Worldwide Rights Reserved

Author Profile

*Rick Schummer is the lead geek at his company White Light Computing, Inc., which is headquartered in southeast Michigan, USA. He prides himself in guiding his customer's Information Technology investment toward success. After hours you might find him creating developer tools that improve developer productivity, or writing articles for his favorite Fox periodicals and user group newsletters. Rick is a co-author of *Deploying Visual FoxPro Solutions*, *MegaFox: 1002 Things You Wanted To Know About Extending Visual FoxPro*, and *1001 Things You Always Wanted to Know About Visual FoxPro*. He is a founding member and Secretary of the Detroit Area Fox User Group (DAFUG) and a regular presenter at user groups in North America. Rick has enjoyed presenting at GLGDW 2000-2003, Essential Fox 2002-2004, VFE DevCon2K2, and scheduled to speak at the Southwest Fox 2004 conference.*

*raschummer@whitelightcomputing.com, rick@rickschummer.com,
<http://www.whitelightcomputing.com> and <http://www.rickschummer.com>*