New VFPX Projects

Doug Hennig
Stonefield Software Inc.
Email: doug@doughennig.com
Corporate Web sites: stonefieldquery.com
stonefieldsoftware.com
Personal Web site : DougHennig.com
Blog: DougHennig.BlogSpot.com
Twitter: DougHennig

This session looks at three new VFPX projects: VFPX Framework, FoxGet, and DeployFox.

VFPX Framework provides base UI classes and commonly used functions such as reading from or writing to INI files and the Window Registry, recursive file operations with a dialog such as copying files and deleting folders, and so on. Not only does VFPX Framework eliminate duplicated functionality for VFPX projects, it can also be used in your own non-VFPX applications.

FoxGet is the VFP equivalent of the NuGet .NET package manager. The idea is that you run FoxGet when you want to add a library to an application. You search for a library you're interested in and if one is found, you can download, install, and add it to your project with a single mouse click.

DeployFox automates the steps required to deploy your applications by providing a customizable list of tasks: copy files, rename files, build a project into an EXE, digitally sign an EXE, run an Inno Setup script to create an installer, upload files to an FTP site, and so on.

Introduction

This session looks at three new VFPX projects: VFPX Framework, FoxGet, and DeployFox.

Thor, GoFish, Project Explorer, and every other VFPX project has a different set of UI classes, subclasses of the VFP base classes. Many projects have common functionality: for example, Thor, FoxGet, and DeployFox all have functions to download and unzip files. Until now, there hasn't been a common shared set of classes and libraries for VFPX projects. Now there is: VFPX Framework. It provides base UI classes and commonly used functions such as reading from or writing to INI files and the Windows Registry, getting a filename from the user using a better dialog than GETFILE(), recursive file operations with a dialog such as copying files and deleting folders, and so on. Not only does VFPX Framework eliminate duplicated functionality for VFPX projects, it can also be used in your own non-VFPX applications.

If you've worked with Visual Studio, you've likely used NuGet, which is a package manager for .NET. The idea is that you can search for libraries you'd like to add to your application, download and install them, and then have them managed (automatically download again if files are missing, update to a new version, etc.). FoxGet is the VFP equivalent of NuGet. The idea is that you run FoxGet when you want to add a library to an application. You search for a library you're interested in and if one is found, you can download, install, and add it to your project with a single mouse click.

Until recently, I used a Microsoft Excel document with a long list of tasks as a checklist for application deployment. These tasks were almost all manual. I've always wanted to automate deployment as much as possible, so I created DeployFox. DeployFox automates the steps required to deploy your applications by providing a customizable list of tasks: copy files, rename files, build a project into an EXE, digitally sign an EXE, run an Inno Setup script to create an installer, upload files to an FTP site, and so on. Now I mostly just open a DeployFox project and run it to deploy an application, saving me lots of time and making an error-free process.

VFPX Framework

VFPX Framework is available at https://github.com/VFPX/VFPXFramework or can be installed using FoxGet (see the **FoxGet** section of this document). It consists of two sets of components:

- Base classes
- Specialty components

Base classes

VFPXBaseLibrary.vcx contains subclasses of most VFP base classes. They are named VFPXBase*class*, where *class* is the base class name, such as VFPBaseTextbox for a textbox. See the documentation in the GitHub repository for detailed information.

Some of the more useful changes from the base appearance/behavior of VFP base classes are:

- Builder: the name of a builder (specified as BuilderName.prg or BuilderLibrary,BuilderClass) to use for the class. For VFPXBaseGrid, this is set to VFPXGridBuilder.prg, which is discussed below.
- BuilderCode: code for a self-container builder; see the notes in the method for instructions to use.
- Enabled_Assign (VFPXBaseContainer and VFPXBasePage): sets the Enabled property of all contained controls to the same value.
- InteractiveChange and ProgrammaticChange: call UpdateControlSource to write the control's value to its control source, then call AnyChange. Put code in AnyChange that should be called on any change (programmatic or interactive) to the control's value.
- RightClick: calls This.ShowMenu, which instantiates an SFMenuShortcutMenu object into the oMenu property (this requires the VFPX OOPMenu project be installed, which is automatically done if you use FoxGet to install VFPX Framework) and calls ShortcutMenu to populate the shortcut menu. See the code in VFPXBaseTextBox.ShortcutMenu for an example of how to populate a shortcut menu.
- RowSource and RowSourceType (VFPXBaseComboBox and VFPXBaseListBox): This.aItems and 5-Array, respectively. If the lRequeryOnInit property is .T. (the default), populate the This.aItems array in Init and then call DODEFAULT() to requery the control.
- lAddNewItemToList (VFPXBaseComboBox): if .T., Style is 1-Dropdown Combo, RowSourceType is 5-Array, and RowSource is This.aItems, values entered by the user that don't exist in the array are automatically added to it by the Valid method.
- lSaveAnchor: set this to .T. before programmatically changing the Top, Left, Height, or Width properties of the control to ensure anchoring is handled properly, then set it to .F. afterward.

Many of the classes use VFPXBaseLibrary.h, which contains some commonly used constants, as their include file.

Specialty components

See the documentation in the GitHub repository for information about the specialty components. The more useful ones are discussed here.

VFPXGridBuilder

Grids can be a pain to set up, especially if you later need to add a column between two existing columns. VFPXGridBuilder provides an easy way to define the columns of a grid by using a format definition.

To use it, put the format definition on the clipboard and invoke the builder. If you're using a VFPXGrid object, right-click the grid and choose Builder. For any other type of grid, select the grid and run VFPXGridBuilder.prg.

The easiest way to copy the format definition is to put it as comments in some method of the grid, such as Init. You then copy the text and invoke the builder.

Here's an example of a format definition:

```
Field     |Width  |Caption    |Alignment  |InputMask  |Format  |ReadOnly  |Control
Invnum    |70     |Invoice #  |           |           |        |.T.       |
Date      |70     |Date       |           |           |        |.T.       |
Name      |*      |Project    |           |           |        |.T.       |
Amount    |60     |Amount     |R          |99,999.99  |        |.T.       |
Paid      |70     |Paid       |           |           |        |          |Checkbox
DatePaid  |70     |Date Paid  |           |           |        |          |
Received  |60     |Received   |R          |99,999.99  |        |          |
```

Here's some information about the format definition:

- It must have a header row like in the example.
- The supported settings are Field (the ControlSource), Width, Caption, Alignment, InputMask, Format, ReadOnly, and Control.
- The order of the settings is unimportant (the setting names are read from the header) and all but Field and Caption are optional.
- Separate columns with tabs and a pipe character.
- Use an empty setting for an unspecified value.
- Use "*" for Width to auto-size a column; that is, size it to the rest of the space after the other columns are sized.
- Alignment can be specified as 0 or L for left, 1 or R for right, and 2 or C for center.
- If Control isn't specified, a Textbox is used. Otherwise, the specified class is used. Currently only Checkbox, CommandButton, and Combobox are supported for Control.

VFPXPersistentForm

Users appreciate a form that opens in the same position and size as it was last time it was open. VFPXPersistentForm in VFPXPersistentForm.vcx provides this ability. Create a form based on VFPXPersistentForm and set the cRegistry property to the Windows Registry location in HKEY_CURRENT_USER to save the window settings (for example, "Software\MyApp"). VFPXPersistentForm sizes and positions the form properly, including ensuring it fits on the correct monitor. For example, if the user has two monitors, the form was opened on the second monitor before, but now only one monitor is available, the form is opened on the correct monitor and sized and positioned appropriately.

VFPXPersistentForm uses two other components, VFPXPersist and SFMonitors.prg, and VFPXRegistry

VFPXDropDownMenuButton

VFPXDropDownMenuButton in VFPXDropDownMenuButton.vcx provides a button with a dropdown menu, sometimes called a "split" button. Drop one on a form, set Picture property of the cmdMain button as desired, fill in the ShortcutMenu method of the VFPXDropDownMenuButton object as necessary, and put code into the ButtonClicked method that executes when the user clicks the button.

For example, DeployFox has a VFPXDropDownMenuButton allowing you to either click the button to open a project or select from a dropdown list of previously opened projects (**Figure 1**).

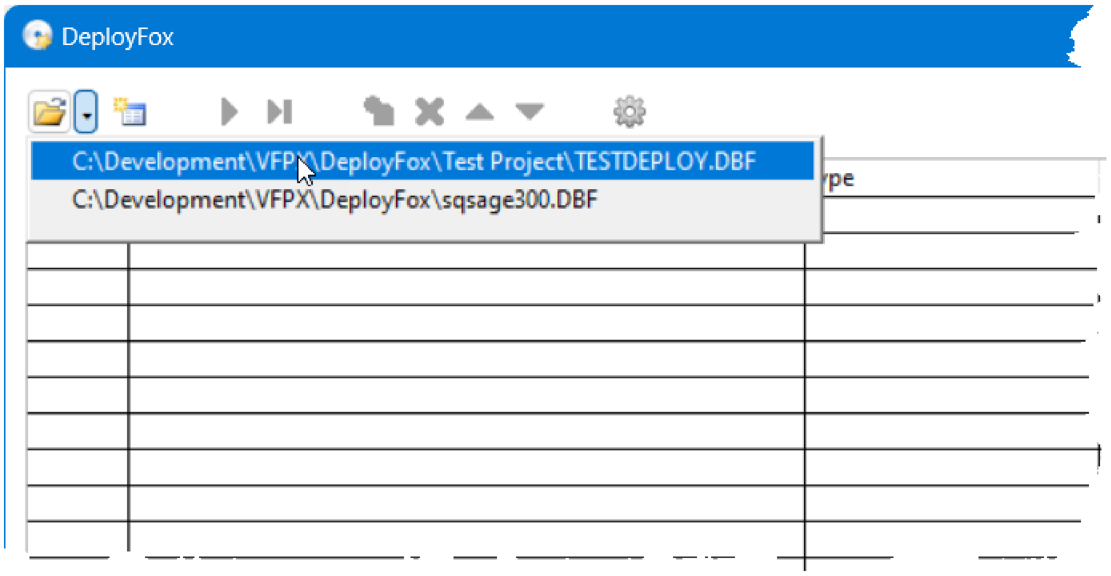

DeployFox
C:\Development\VFPX\DeployFox\Test Project\TESTDEPLOY.DBF
C:\Development\VFPX\DeployFox\sqsage300.DBF
ype

Figure 1. VFPXDropDownMenuButton provides a split button.

Move, copy, delete, or rename files and folders

VFP has commands to do all these operations but:

- No dialog is displayed if there are a lot of files (or large files) to move or copy.
- They aren't recursive; that is, they don't support a folder and its subdirectories.
- The VFP RD command throws an error if the folder isn't empty.

FileOperation.prg, adapted from code written by Sergey Berezniker, is a better way to perform these operations. Pass it up to six parameters:

- Source: the file or folder to copy, move, delete, or rename. Required.
- Destination: the file or folder to copy or move the source to. Optional for a delete operation, required for the others.

- Operation: the action to perform: “move,” “copy,” “delete,” or “rename.” Required.
- UserCanceled: optionally passed a variable by reference; upon return, it contains .T. if the user canceled the operation.
- Files only: optionally pass .T. to process file not folders.
- Quiet: optionally pass .T. to not display a dialog.

FileOperation returns .T. if the operation succeeded or the user canceled. It uses ClsHeap.prg.

File dialogs

The VFP GETFILE() and PUTFILE() functions have a *lot* of shortcomings. Here are just a few:

- They return the path in uppercase.
- They don’t support multi-file selection.
- They are older dialogs that don’t support all the features of newer ones.
- They don’t support a default folder.

The VFPXCommonDialog class in VFPXCommonDialog.vcx is a customized version of the _ComDlg class in the FFC _system.vcx (for example, it’s easier to call because you don’t have to pass parameters by reference). Because it uses the standard Windows file dialog, the dialog’s appearance and behavior always matches the version of Windows it’s running on.

GetFileName.prg is a wrapper for VFPXCommonDialog. Pass it up to five parameters:

- File extensions: the same value as the extensions parameter for GETFILE(). Optional: if it isn’t specified, all file types are allowed.
- Default file path: optional. If passed the folder for the file is used as the default folder and the filename textbox in the dialog is filled in with the filename.
- Dialog title caption: optional; if not specified, the caption is “Open” or “Save,” depending on the value of the next parameter.
- .T. for a Save dialog, .F. or not specified for an Open dialog.
- .T. to allow multiple files to be selected, .F. or not specified for a single file.

It returns a comma-separated list of files the user chose or blank if they clicked Cancel.

Reading from and writing to INI files

ReadINI.prg and WriteINI.prg allow you to read from and write to INI files.

ReadINI accepts these parameters:

- tcINIFile: the INI file to look in
- tcSection: the section to look for
- tuEntry: the entry to look for (pass 0 and taEntries to enumerate all entries in the section)
- tcDefault: the default value to use if the entry isn't found (optional: an empty string is used if not passed)
- taEntries: an array (passed by reference) to hold all entries in the section (only needed if tuEntry is 0)

If tuEntry is a string (the entry), ReadINI returns the value of the entry (if it begins with 0x, indicating it's stored as hexBinary, it's converted back to a normal string) or tcDefault if the entry isn't found. If tuEntry is 0, ReadINI returns the number of entries in the array and the array is filled with the names of the entries.

WriteINI accepts these parameters:

- tcINIFile: the INI file to look in
- tcSection: the section to look for
- tuEntry: the entry to look for (pass NULL to remove the section)
- tuValue: the value to store (pass NULL to remove the entry)

If the value contains any binary characters, it's written out as "0x" + the value converted to hexBinary because binary values may not be read in correctly. Logical values are converted to Y or N and other non-string values are converted to strings. WriteINI returns .T. if the value was written to the INI file.

Reading from and writing to the Windows Registry

VFPXRegistry in VFPXRegistry.vcx provides methods to read from and write to the Windows Registry. It uses VFPXRegistry.h, which includes VFPXBaseLibrary.h.

You may be wondering why not just use the FoxPro Foundation Classes (FFC) _Registry.vcx. There are several advantages of VFPXRegistry over _Registry, but the main ones are that you don't have to pass parameters by reference and it supports types other than just strings, including binary values, DWORD, and multi-string values.

Many of the VFPXRegistry methods accept a hive parameter, the section of the Registry such as HKEY_CURRENT_USER. Pass one of the cnHKEY constants in VFPXRegistry.h, such as cnHKEY_LOCAL_MACHINE; the value of the nMainKey property, which defaults to cnHKEY_CURRENT_USER, is used if it isn't passed. If you intend to issue several calls to methods for a particular hive, set nMainKey to the desired hive and you can then omit the hive parameter in those calls.

Many methods also accept a value name. If it isn't specified, the default value for the key is used.

There are several methods in VFPXRegistry but the main ones are:

- GetKey: pass it the key, the value name (optional: see above), the default value to return if the key or value name doesn't exist (optional: an empty string is used if not passed), the hive (optional: see above), and .T. to look in the 64-bit version of the hive. GetKey returns the value of the specified value name, or the default value if the key or value name doesn't exist.
- SetKey: pass it the key, the value name (optional: see above), the value to write, the hive (optional: see above), and the type of value to write to (one of the cnREG constants in VFPXRegistry.h, such as cnREG_DWORD for a 32-bit number; optional: cnREG_SZ, meaning a string value, is used if not passed). SetKey returns .T. if the value was written. Non-character values are written as strings (logical as Y or N) if the value type is cnREG_SZ or not passed.
- DeleteKeyValue: pass it the key, the value name (optional: see above), and the hive (optional: see above) to delete the specified value.
- EnumerateKeyValues: pass it the key, an array (passed by reference), and the hive (optional: see above) to fill the specified array with the names of the values of the specified key in the first column and their values in the second. EnumerateKeys returns the number of entries in the array.

Uploading and downloading files

The VFPXInternet class in VFPXInternet.prg has methods to upload and download files. Pass DownloadFile the path of the remote file to download, the path of the local file to download to, the server, and the user name and password (both optional) to connect to the server. UploadFile accepts the same set of parameters. Both methods return .T. if they succeed and set cErrorMessage to the error message if they fail.

VFPXInternet uses curl.exe, which is included with Windows 10 version 1803 or later. Otherwise, you can download it from https://curl.se/windows.

Zipping and unzipping files

Craig Boyd's VFPCompression.fll is a common way for VFP developers to zip and unzip files. While it's a great tool, it also has a bug that prevents it from unzipping all of the files in a zip file under some conditions (I don't know what the conditions are but at least one VFPX project fails to unzip properly when installed using Thor's Check for Updates).

VFPXZip.prg uses the Windows Shell to zip and unzip files; if it fails (usually due to disabling the Windows Shell for security reasons), PowerShell is used instead. These mechanisms aren't as flexible as VFPCompression.fll but are more reliable.

To zip files to a new or existing zip file, pass the Zip method a comma-delimited list of file paths, the path for the zip file, and .T. to overwrite any existing file or .F. to update any existing file. To unzip a zip file, pass the Unzip method the path for the zip file and the folder to unzip the files into. If anything goes wrong, the methods return .F. and cErrorMessage contains the error message. VFPXZip.prg uses several other components of VFPX Framework: GetProperFileCase.prg, ExecuteCommand.prg, and API_AppRun.prg.

The other projects we'll discuss, FoxGet and DeployFox, both use VFPX Framework components.

FoxGet

Thor (https://github.com/VFPX/Thor) is a very popular tool amongst VFP developers. It has a Check for Updates (CFU) feature that can install and update projects. While this works very well for "tools," projects that are used within the VFP IDE like GoFish, it is less suitable for installing "components," projects that add features to your applications:

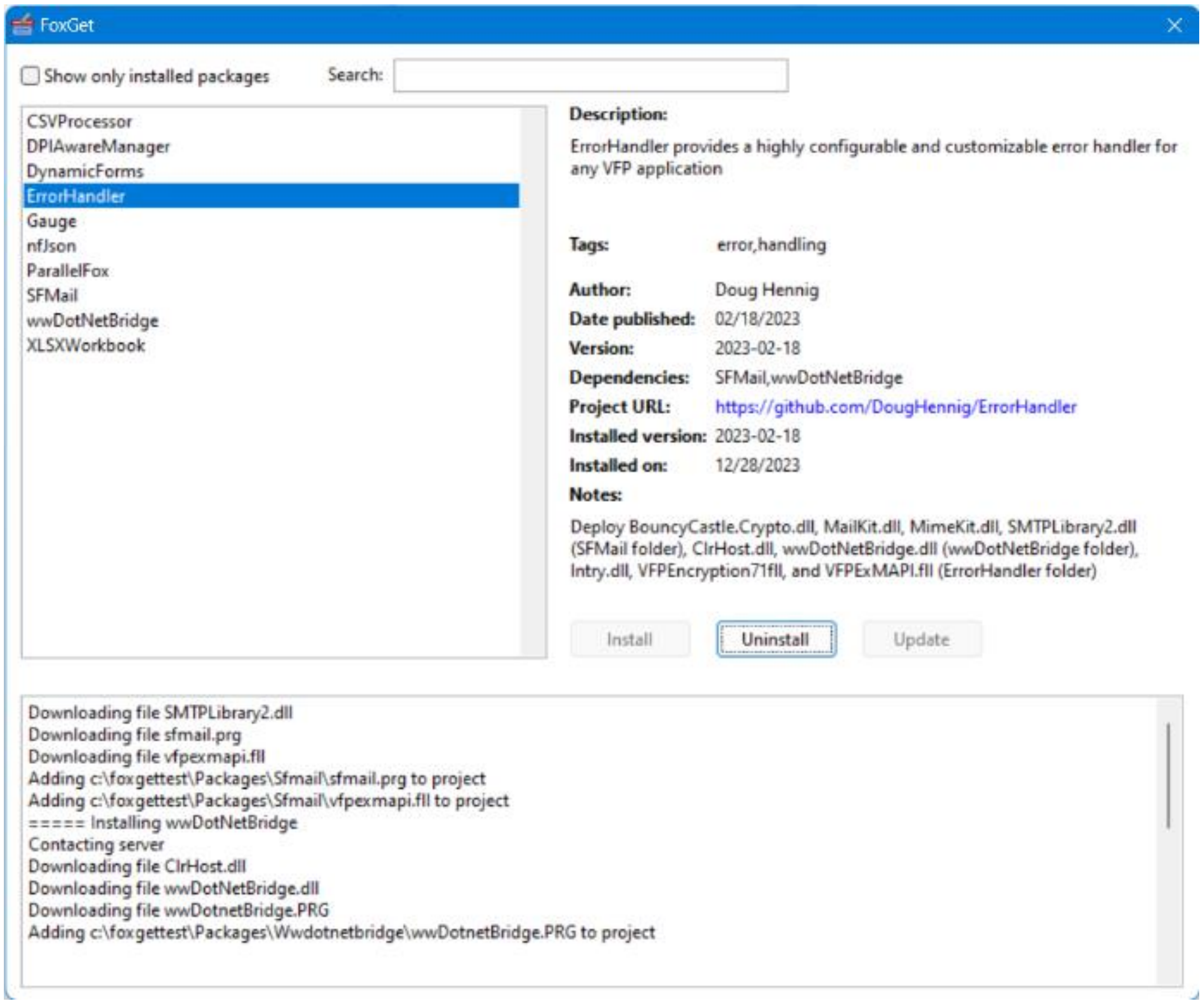
- Thor installs projects in a subdirectory of its own folder rather than under your application, making source code control and pathing trickier.
- Thor doesn't list all projects, only those the project manager has configured to work with Thor CFU. FoxGet doesn't list all projects either, but projects can be expanded without updating the repository of the project.

For these reasons, FoxGet is more suited to adding other components to your applications. Note: in this document, "package" is another named for a component or a library.

FoxGet is available at https://github.com/doughennig/foxget or you can use Thor Check for Updates to install it.

Using FoxGet

To use FoxGet to add components to an application, open the project for the application and DO FoxGet.app in the FoxGet folder or, if you used Thor to install it, choose FoxGet from the Thor Tools, Applications menu. The dialog shown in **Figure 2** appears.



CSVProcessor
DPIAwareManager
DynamicForms
ErrorHandler
Gauge
nfJson
ParallelFox
SFMail
wwDotNetBridge
XLSXWorkbook

Description:

ErrorHandler provides a highly configurable and customizable error handler for any VFP application

Tags:	error,handling
Author:	Doug Hennig
Date published:	02/18/2023
Version:	2023-02-18
Dependencies:	SFMail,wwDotNetBridge
Project URL:	https://github.com/DougHennig/ErrorHandler
Installed version:	2023-02-18
Installed on:	12/28/2023

Notes:

Deploy BouncyCastle.Crypto.dll, MailKit.dll, MimeKit.dll, SMTPLibrary2.dll (SFMail folder), ClrHost.dll, wwDotNetBridge.dll (wwDotNetBridge folder), Intry.dll, VFPEncryption71fll, and VFPExMAPI.fll (ErrorHandler folder)

```
Downloading file SMTPLibrary2.dll
Downloading file sfmail.prg
Downloading file vfpexmapi.fll
Adding c:\foxgettest\Packages\Sfmail\sfmail.prg to project
Adding c:\foxgettest\Packages\Sfmail\vfpexmapi.fll to project
===== Installing wwDotNetBridge
Contacting server
Downloading file ClrHost.dll
Downloading file wwDotNetBridge.dll
Downloading file wwDotnetBridge.PRG
Adding c:\foxgettest\Packages\Wwdotnetbridge\wwDotnetBridge.PRG to project
```

Figure 2. FoxGet allows you to search for and install packages.

Select a package to see information about it on the right, including the version and date the package was installed in the project if it was installed. Click the link for *Project URL* to go to the home URL for the package.

You can search for a package by name, tag, or description by typing in the Search textbox. To show only packages installed in the current project, turn on *Show only installed packages*.

To install the selected package, click Install; that button is disabled if the package has already been installed. After a moment, you should see that some files were added to the project and there's a Packages subdirectory of the project folder containing Packages.dbf and the downloaded files in a subdirectory for the component. The package subdirectory also contains a file named *Package*/Installer.prg, which is used to uninstall the package.

You may wonder why FoxGet puts the library into a subdirectory of the Packages subdirectory of the project folder rather than in a common location other applications could reference. There are several reasons:

- That's the way NuGet works.
- If your application is in source control (such as Git), it isn't easy to include paths outside the application path in the repository.
- You may want to use different versions of a library in different applications, especially if how you call the library changes between versions.

Since packages go in their own folders, you'll need to set a path to those folders if you run the application in the VFP IDE.

To uninstall the selected package, click the Uninstall button. The files added to the project by the installer are removed from the project, the package folder in the Packages subdirectory is deleted, and Packages.dbf is updated.

If there's a newer version of the package available, the Update button is enabled. Clicking it uninstalls the package then installs the new version.

Dependencies

Some projects depend on other projects. For example, ErrorHandler (https://github.com/DougHennig/ErrorHandler) uses SFMail (https://github.com/DougHennig/SFMail), which itself uses wwDotNetBridge (https://github.com/RickStrahl/wwDotnetBridge). FoxGetPackages.dbf, which contains information about each package, has a Dependent column containing the names of other packages a package is dependent on. When you install a package, all dependencies are also installed (any that are already installed are reinstalled). When you uninstall a package, dependencies may be uninstalled as well, as long as no other packages depend upon them and they weren't installed as a standalone package.

Note that dependencies go in their own package folders, so you'll need to set a path to those folders if you run the application in the VFP IDE.

Creating a package installer

If you're interested in writing your own installer, see the FoxGet documentation.

DeployFox

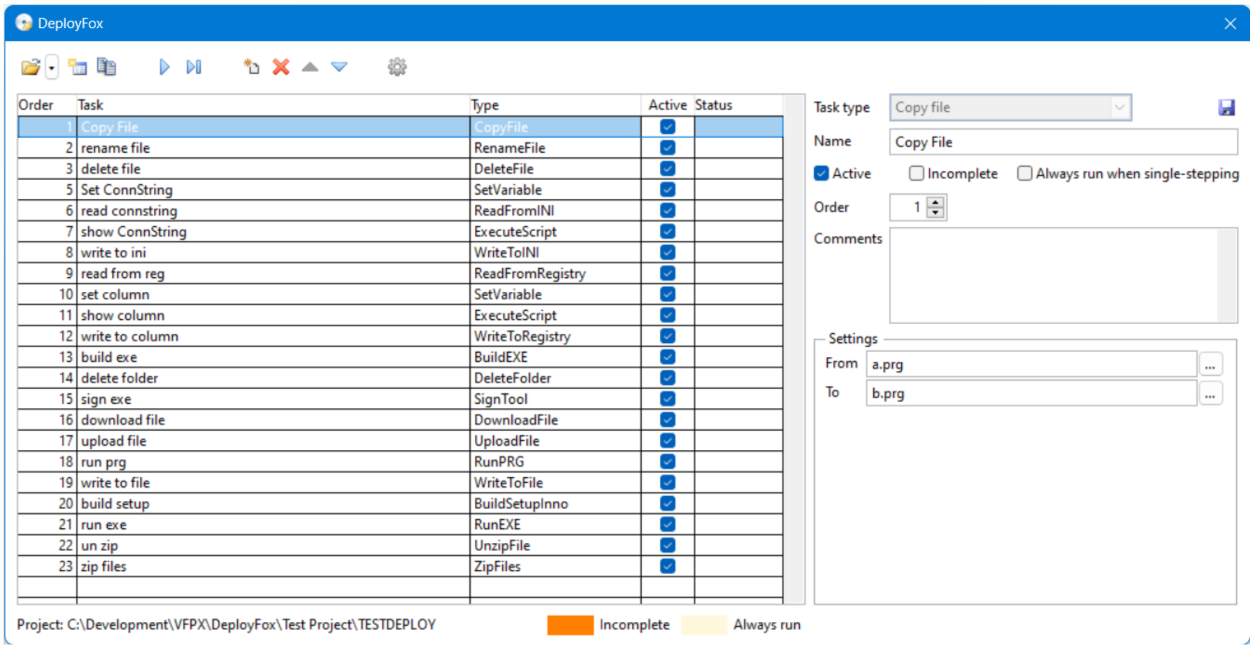
Tracy Pearson did a presentation at Southwest Fox 2019 titled "VFP DevOps: Implementing an Automated Build for a Complex Release of a Vertical Market Application" (https://swfox.net/2019/SessionsSWFOX.aspx#DevOps_Implementing_an_Automated_Build) that explains in detail the benefits of automating deployment.

Deploying an application consists of a set of tasks. In DeployFox, sets of tasks are called a project and are stored in a table located wherever you wish (usually a subdirectory of the folder for the application to deploy). DeployFox supports a lot of types of tasks, such as copying files, renaming files, uploading and downloading files, and so on.

DeployFox is available at https://github.com/doughennig/deployfox or you can use Thor Check for Updates to install it.

DeployFox UI

To run DeployFox, DO DeployFox.app in the DeployFox folder or, if you used Thor to install it, choose DeployFox from the Thor Tools, Applications menu. The dialog shown in **Figure 3** displays.



Order	Task	Type	Active	Status
1	Copy File	CopyFile	✓	
2	rename file	RenameFile	✓	
3	delete file	DeleteFile	✓	
5	Set ConnString	SetVariable	✓	
6	read connstring	ReadFromINI	✓	
7	show ConnString	ExecuteScript	✓	
8	write to ini	WriteToINI	✓	
9	read from reg	ReadFromRegistry	✓	
10	set column	SetVariable	✓	
11	show column	ExecuteScript	✓	
12	write to column	WriteToRegistry	✓	
13	build exe	BuildEXE	✓	
14	delete folder	DeleteFolder	✓	
15	sign exe	SignTool	✓	
16	download file	DownloadFile	✓	
17	upload file	UploadFile	✓	
18	run prg	RunPRG	✓	
19	write to file	WriteToFile	✓	
20	build setup	BuildSetupInno	✓	
21	run exe	RunEXE	✓	
22	un zip	UnzipFile	✓	
23	zip files	ZipFiles	✓	

Task type: Copy file
Name: Copy File
[✓ Active] Incomplete Always run when single-stepping
Order: 1
Comments:
Settings
From: a.prg
To: b.prg

Project: C:\Development\VFPX\DeployFox\Test Project\TESTDEPLOY 🟧 Incomplete ▨ Always run

Figure 3. DeployFox automates the steps in deploying an application.

The dialog consists of four sections: a toolbar at the top, a list of tasks in the middle, properties about the selected task at the right, and a status bar at the bottom.

The toolbar has the following functions:

- : opens a DeployFox project. Click the dropdown button to display a list of previous opened projects.
- : creates a new DeployFox project in the folder you specify and with the name you choose.
- : clones the current project. This is useful if you need another project similar to the current one but with some differences.
- : runs (executes the tasks in) the project.

- ⏯: debugs the tasks in the project. Debugging is similar to running but starts from the selected task rather than the first one and after running a task asks if you want to run the next one or stop the execution. In other words, it single-steps through the tasks.
- 🗋: adds a task to the project. It's added as the next step but can be moved up to an earlier step.
- ✖: removes the selected task.
- ▲ and ▼: moves the selected task up and down in the list and renumbers the steps accordingly.
- ⚙: displays the Settings dialog.

The list of tasks shows the order (step number), task name, task type, whether it's active, and the status of the last run (success or failure). You can make a task active or inactive by clicking the checkbox in the grid or in the properties for the selected task at the right (in the latter case, you have to click Save to save the change).

The properties for the selected task displayed at the right are:

- *Task type*: this can only be changed when a task is added; after it's saved, this is disabled. To change the type of a task, delete and re-add it.
- *Name*: the name of the task.
- *Active*: inactive tasks are skipped when the project is run.
- *Incomplete*: turn this on for tasks you're working on but haven't finished yet. Incomplete tasks aren't executed.
- *Always run when single-stepping*: some tasks are dependent on others. For example, some tasks use a variable so require that the task that sets the variable's value execute before the task does (see the **Expressions and variables** section for details on variables). Turn this setting on for those tasks that always execute first before single-stepped tasks do.
- *Order*: the task order. You can either change the order by entering a different value or clicking the ▲ and ▼ buttons in the toolbar.
- *Comments*: comments or notes about the task.
- *Settings*: the settings for a task are specific to the task type and are discussed below.

The status bar displays the path for the open project file and a color chart showing the colors used in the grid for the status of the tasks.

Task types

The types of tasks DeployFox supports are stored in a table, TaskTypes.dbf, along with the name of the PRG containing a class definition for the task (the class name must be the name of the task type, such as DeleteFile; all pre-defined tasks types are contained in Tasks.prg) and the VCX containing the UI for the settings for the task type (the class name must be *TaskType*UI, such as DeleteFileUI; the UI for all pre-defined tasks types are contained in TaskUI.vcx). If you want to create your own task types, subclass TaskBase in Tasks.prg into your own PRG, subclass TaskUIBase in TaskUI.vcx into your own VCX, and add a record for the task type to MyTaskTypes.dbf.

DeployFox comes with the following task types.

CopyFile

Copies a file. The settings are the path for the file to copy (*From*) and the destination path (*To*).

DeleteFile

Deletes a file. The only setting is the path for the file.

RenameFile

Renames a file. The settings are the path for the file to rename (*From*) and the new name (*To*).

DeleteFolder

Deletes a folder and all files in it and all subdirectories. The only setting is the path for the folder.

WriteToFile

Writes to a file. The settings are the path for the file (which doesn't have to exist), the text to write, and whether the text should be appended to the file or overwrite any existing content.

ZipFiles

Compresses one or more files into a ZIP file. The settings are the path for the ZIP file (which doesn't have to exist), a list of files to compress, and whether to create a new ZIP file or update an existing one.

UnzipFile

Extracts the files in a ZIP file. The settings are the path for the ZIP file and the folder to extract the files to.

DownloadFile

Downloads a file. The settings are the server, username, and password to connect to the server (if necessary), the path of the local file to download to, and the path of the remote file.

UploadFile

Uploads a file. The settings are the server, username, and password to connect to the server (if necessary), the path of the local file to upload, and the path of the remote file.

RunPRG

Executes a PRG. The settings are the path for the PRG file and any parameters to pass to it. Put quotes around any string parameters.

RunEXE

Executes an EXE. The settings are the path for the PRG file, any parameters to pass to it, and the window mode (Normal, Hidden, Minimized, or Maximized). Put quotes around any parameters containing spaces or other illegal command line characters.

RunBAT

Executes a Window batch (BAT) file. The settings are the path for the BAT file.

ExecutePSScript

Executes a PowerShell script (PS1) file. The settings are the path for the PS1 file.

ExecuteScript

Executes VFP code. The settings are the code to execute.

ReadFromINI

Reads a value from an INI file into a variable (see the **Expressions and variables** section for details on variables). The settings are the path of the INI file, the section and item to read from, and the name of the variable to store the value to.

WriteToINI

Writes a value to an INI file. The settings are the path of the INI file, the section and item to write to, and the value to write.

ReadFromRegistry

Reads a value from the Windows Registry into a variable. The settings are the hive (such as HKEY_CURRENT_USER), key, and setting of the Registry entry and the name of the variable to store the value to.

WriteToRegistry

Writes a value to the Windows Registry. The settings are the hive (such as HKEY_CURRENT_USER), key, and setting of the Registry entry, the value to write, and the value type (String, String with Unexpanded Environment Variables, and 32-bit Number)

SetVariable

Saves a value to a variable, creating the variable if necessary. The settings are the variable name, the value, and whether the value is encrypted or not. Variables are discussed in the next section.

BuildEXE

Builds an EXE from a VFP project. The settings are the path for the project, the path for the EXE, and whether to recompile all files.

SignTool

Digitally signs a file. The settings are the path for the file to sign and the description to apply. This task uses SignTool.exe, which is included with DeployFox, and the settings stored in the Options dialog (discussed below).

BuildSetupInno

Builds a setup executable from an Inno Setup script file. The settings are the path for the script file. Inno Setup, which you can download from https://jrsoftware.org/isinfo.php, must be installed.

Expressions and variables

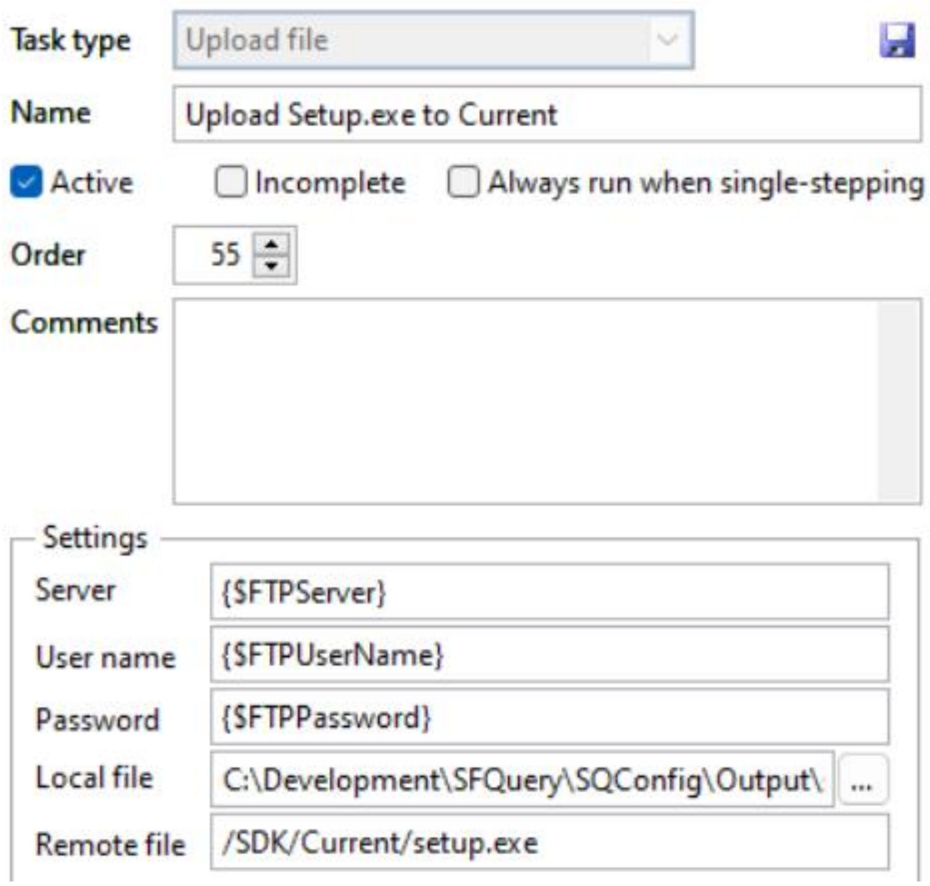
Most settings can either be a literal value (such as "C:\SomePath\SomeFile.png") or an expression. To distinguish them, an expression is surrounded with curly braces; for example, "{fullpath('SomeFile.png')}."

DeployFox allows you to define variables using a SetVariable task or the SetVariable function in a script executed with an ExecuteScript task. For example, if your DeployFox project has several tasks to upload files, you likely don't want to hard-code the server, username, and password in every task, so you can define variables to contain those values and then use the variables in the tasks. **Table 1** shows three sample tasks that define such variables.

Table 1. Tasks defining variables for file upload settings.

Task Name	Variable Name	Value	Encrypted
Set FTPServer	FTPServer	MyServer	No
Set FTPUserName	FTPUserName	MyUserName	No
Set FTPPassword	FTPPassword	MyPassword	Yes

Variables are specified in an expression using a "$" prefix. **Figure 4** shows how the upload variables are used in a task.


Task type: Upload file

Name: Upload Setup.exe to Current

[✓ Active] Incomplete Always run when single-stepping

Order: 55

Comments:

Settings

Server	{$FTPServer}
User name	{$FTPUserName}
Password	{$FTPPassword}
Local file	C:\Development\SFQuery\SQConfig\Output\
Remote file	/SDK/Current/setup.exe

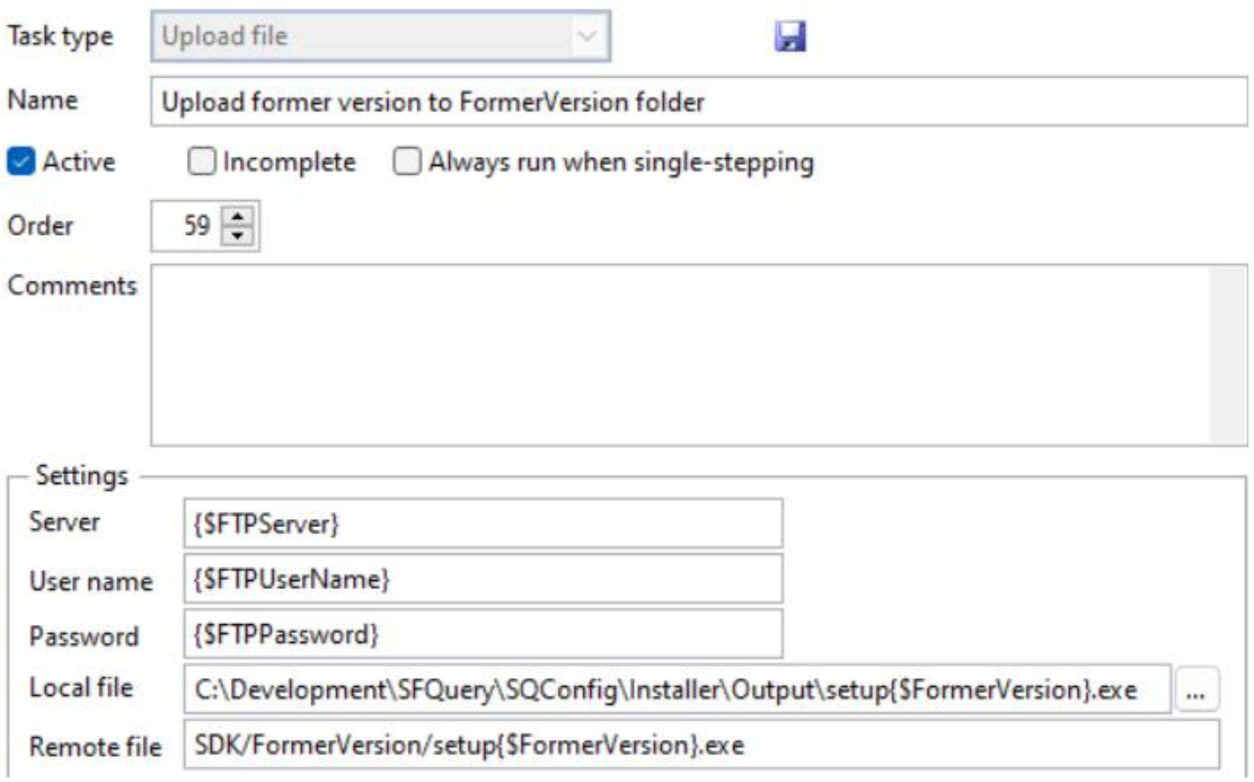
Figure 4. Variables are specified in an expression using a "$" prefix.

The ReadFromINI and ReadFromRegistry task types save the read values to a variable. The variable must be defined using SetVariable first.

You can also define or assign a value to a variable using the SetVariable function. The following code used in an ExecuteScript task runs gets the version number for the most recent Setup*.exe file (such as Setup0001.exe, Setup0002.exe, and so on) and assigns that value to the FormerVersion variable.

```
adir(laFiles, 'C:\Development\SFQuery\SQConfig\Installer\Output\Setup*.exe')
asort(laFiles, 3, -1, 1)
SetVariable('FormerVersion', strextract(laFiles[1, 1], 'setup', '.exe', 1, 1))
```

The task shown in **Figure 5** uses the FormerVersion variable to upload the former version of an installer to the FormerVersion folder of a web site.



Task type: Upload file

Name: Upload former version to FormerVersion folder

[✓ Active] Incomplete Always run when single-stepping

Order: 59

Comments:

Settings

Server: {$FTPServer}

User name: {$FTPUserName}

Password: {$FTPPassword}

Local file: C:\Development\SFQuery\SQConfig\Installer\Output\setup{$FormerVersion}.exe

Remote file: SDK/FormerVersion/setup{$FormerVersion}.exe

Figure 5. The FormerVersion variable specifies which file to upload.

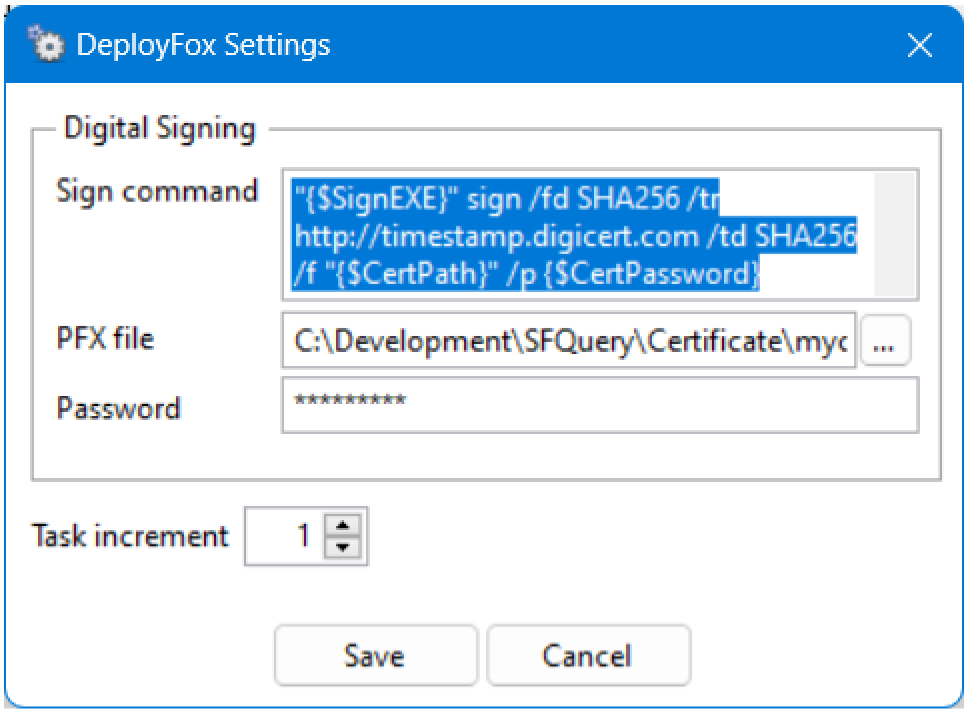
Typically, you'll turn on *Always run when single-stepping* for tasks that assign values to variables so those variables exist and contain values when tasks that depend on them are run.

There are several built-in variables:

- $AppPath: the DeployFox folder (including trailing backslash).
- $BuildEXEWithInno: the command line to build a setup executable using Inno Setup.
- $CertPath: the path to the digital certificate (comes from the *PFX file* setting; see the next section).
- $CertPassword: the password for the digital certificate used to sign an EXE (comes from the *Password* setting; see the next section).
- $ProjectPath: the folder for the open project (including trailing backslash).
- $SignCommand: the command to sign an EXE (comes from the *Sign command* setting; see the next section).
- $SignEXE: the path to SignTool.exe, used to digitally sign an EXE.

Settings

Click the Settings button in the toolbar to display the DeployFox Settings dialog (**Figure 6**).



Sign command: "{$SignEXE}" sign /fd SHA256 /tr http://timestamp.digicert.com /td SHA256 /f "{$CertPath}" /p {$CertPassword}

PFX file: C:\Development\SFQuery\Certificate\myc

Password: **********

Task increment: 1

Figure 6. The DeployFox Settings dialog.

There are three settings related to digitally signing an EXE:

- *Sign command*: the command used to sign an EXE. The default uses SignTool (the path to which is in the $SignEXE variable) to sign the EXE using the digital certificate specified in the *PFX file* setting.
- *PFX file*: the path to the digital certificate.
- *Password*: the password for the digital certificate.

Note: the mechanism used for digital signing requires a local PFX file and currently doesn't support the new mechanism that uses a dongle.

Task increment specifies how much to increment or decrement a task order by when you click the up and down buttons.

Summary

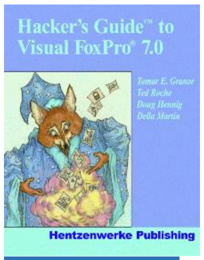
VFPX Framework can be a starting point for new VFPX projects, eliminates duplicated functionality, and can also be used in your own non-VFPX applications. FoxGet provides a fast way to add any library to your application. DeployFox automates application deployment as much as possible, making it a faster and less error-prone mechanism. I look forward to your feedback!

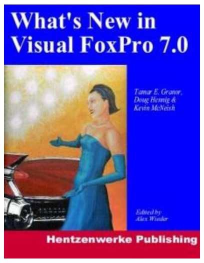
Biography

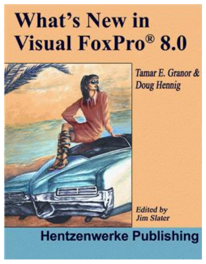
Doug Hennig is a partner with Stonefield Software Inc. He is the author of the award-winning Stonefield Query; the award-winning Stonefield Database Toolkit (SDT) (now open source); the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna. He also created several VFPX projects, including Project Explorer, OOP Menu, OOP Reports, and SFMail.

Doug is co-author of *VFPX: Open Source Treasure for the VFP Developer*, *Making Sense of Sedna and SP2*, *Visual FoxPro Best Practices For The Next Ten Years*, the *What's New in Visual FoxPro* series, and *Hacker's Guide to Visual FoxPro 7.0* (now open source). He was the technical editor of *Hacker's Guide to Visual FoxPro 6.0* and *The Fundamentals*. Doug wrote hundreds of articles in 20 years for *FoxRockX*, FoxTalk, FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe magazines.

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the Southwest Fox and Virtual Fox Fest conferences. He is one of the administrators for the VFPX VFP community extensions Web site. He was a Microsoft Most Valuable Professional (MVP) from 1996 through 2011. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award.






What's New in Visual FoxPro® 8.0
Tamar E. Granor & Doug Hennig
Hentzenwerke Publishing