Anatomy of a VFPX Project

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

Even if you haven't considered contributing to a VFPX project (and you should), you may be curious about what goes into such a project behind the scenes. This document delves into all aspects of a VFPX project, including organization and deployment.

Introduction

Since its inception, VFPX has been an incredible resource for VFP developers. It provides free, production-quality code from some of the best VFP developers on the planet over its more than 130 separate projects. I personally use more than 30 of these projects in my development environment and applications.

Maybe you have some code, or even just an idea, for a project you think other developers could use. Maybe you have some ideas to improve an existing project. Maybe you're just interested in how VFPX works behind the scenes. This document discusses how VFPX is organized, how to start a new project, and how to contribute to an existing project. It also looks at some best practices for coding and documenting projects.

Introduction to VFPX

If you're new to VFPX, this section is for you.

VFPX is located on GitHub, a Microsoft-owned site for Git repositories. Git is a distributed version control system, or DVCS, that's extremely popular with developers world-wide. This document isn't intended to be exhaustive documentation for Git or GitHub; there are many sources of information on both, including https://docs.github.com. Fernando Bozzo, who has contributed numerous VFPX projects, has some excellent Git information at https://github.com/fdbozzo/git_training. Scott Hanselman's blog post "The Squishy Side of Open Source" (https://www.hanselman.com/blog/the-squishy-side-of-open-source) has some good ideas for those new to open source.

Let's start with GitHub, since you don't actually need Git to work with any VFPX project.

GitHub

VFPX is located at https://github.com/vfpx. However, there isn't much to see there: just a list of repositories in reverse chronological order by last update. Also, it doesn't show all VFPX projects, only those under the VFPX GitHub account (we'll see later that many projects are not located there). A better URL is https://vfpx.github.io, the GitHub Pages site for VFPX (GitHub Pages provide a web site for a project). However, the best URL is http://vfpx.org, which always redirects to the current location of VFPX (not that the location will be changing any time in the foreseeable future).

Thor, a VFPX tool (https://github.com/VFPX/Thor) for managing add-ons in the VFP IDE, provides a way to get to the VFPX home page right from within VFP: the *VFPX Home Page* item in the Thor menu.

Whichever way you get to it, the VFPX home page (**Figure 1**) is the first thing you see.

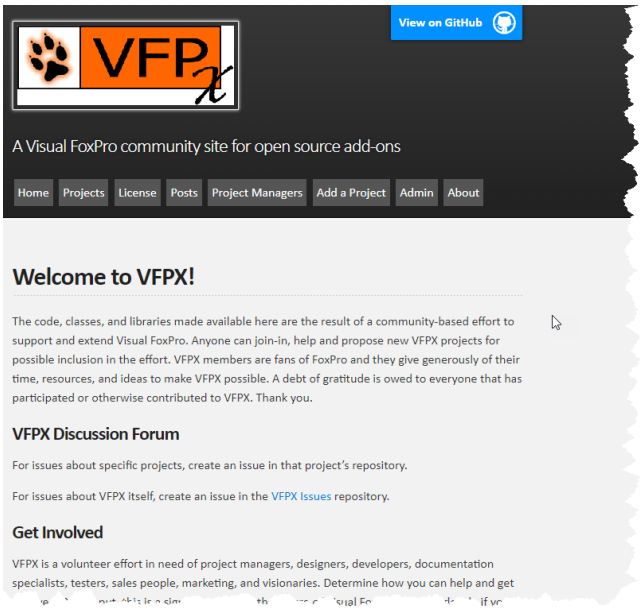
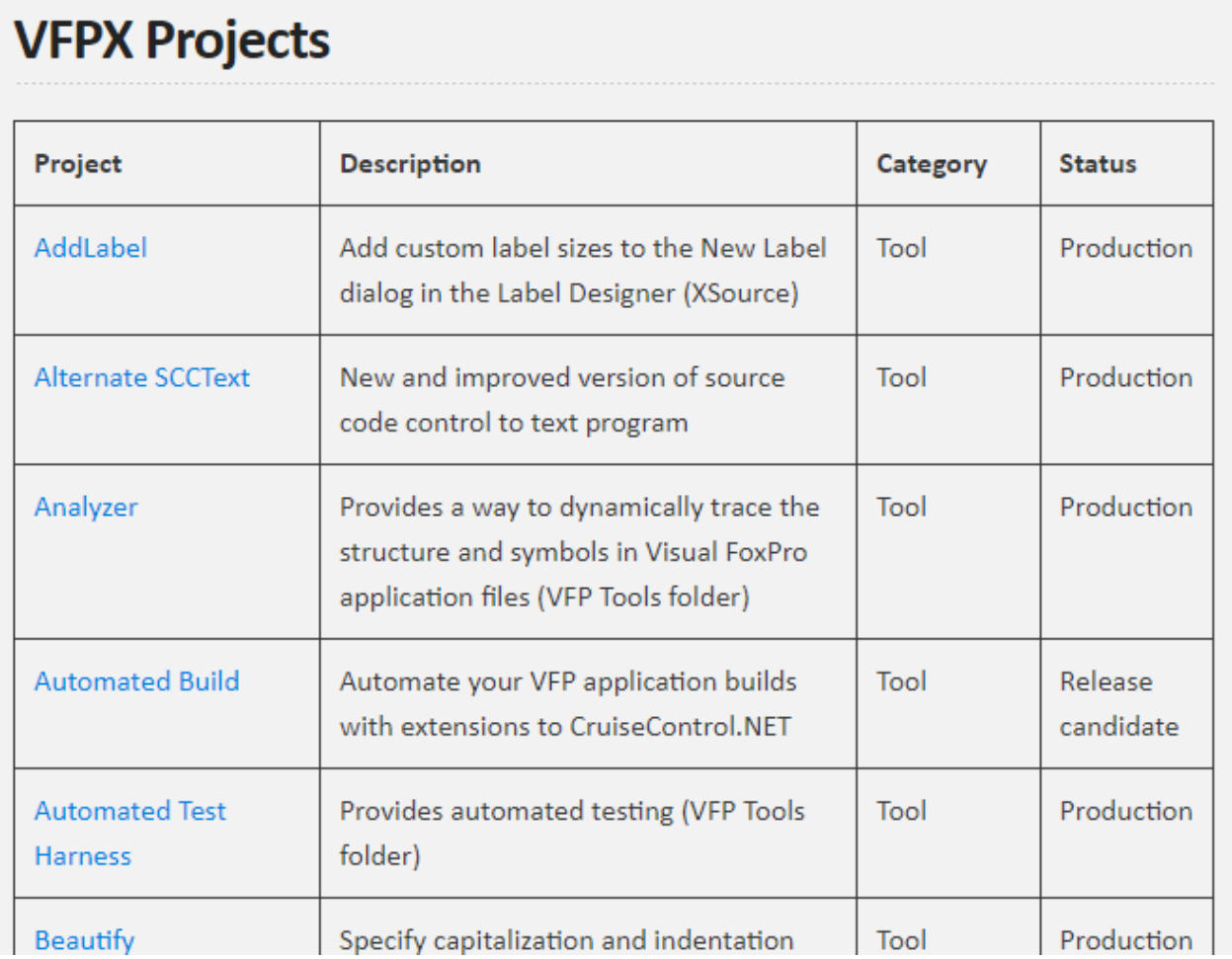


Figure 1. The VFPX home page.

The home page describes what VFPX is, how to get involved, and how to promote it. It also has a menu with the following items:

- Home: a link to the home page.
- Projects: a list of the projects including links to their repositories.
- License: the license all VFPX adhere to unless otherwise stated.
- Posts: news items about VFPX. There's an RSS feed for this page (the "subscribe via RSS" link at the bottom) so you'll be notified when posts are added.
- Project Managers: information for project managers.
- Add a Project: information about how to add a project to VFPX.
- Admin: information for the VFPX administrators (our own documentation).
- About: lists the VFPX admins.

The Projects page (**Figure 2**) is the most used page: it lists the projects alphabetically and shows a brief description, category ("Tool" means a utility that adds features to the VFP Interactive Development Environment, or IDE, and "Component" means a utility that adds features to your VFP applications), and status (Production, Release Candidate, Beta, Alpha, and Planning). The page also lists some other VFP open-source projects that aren't considered to be part of VFPX. Click the name link to navigate to the repository for the project.



VFPX Projects

Project	Description	Category	Status
AddLabel	Add custom label sizes to the New Label dialog in the Label Designer (XSource)	Tool	Production
Alternate SCCText	New and improved version of source code control to text program	Tool	Production
Analyzer	Provides a way to dynamically trace the structure and symbols in Visual FoxPro application files (VFP Tools folder)	Tool	Production
Automated Build	Automate your VFP application builds with extensions to CruiseControl.NET	Tool	Release candidate
Automated Test Harness	Provides automated testing (VFP Tools folder)	Tool	Production
Beautify	Specify capitalization and indentation	Tool	Production

Figure 2. The VFPX Projects page lists each project by name.

Thor provides a way to get to a specific project's home page right from within VFP: choose the *Project Home Pages* item in the Thor menu and click the desired project in the dialog that appears. Note that only projects Thor knows about are listed in that dialog.

Some projects are hosted on GitHub under the VFPX organization (https://github.com/VFPX/*ProjectName*) while others are under the author's repository (https://github.com/*Author*/*ProjectName*).

Figure 3 shows a typical VFPX project repository, in this case the one for the XLXSWorkbook project. As this document is not intended to be complete GitHub documentation, I won't go through everything, just the most common features.

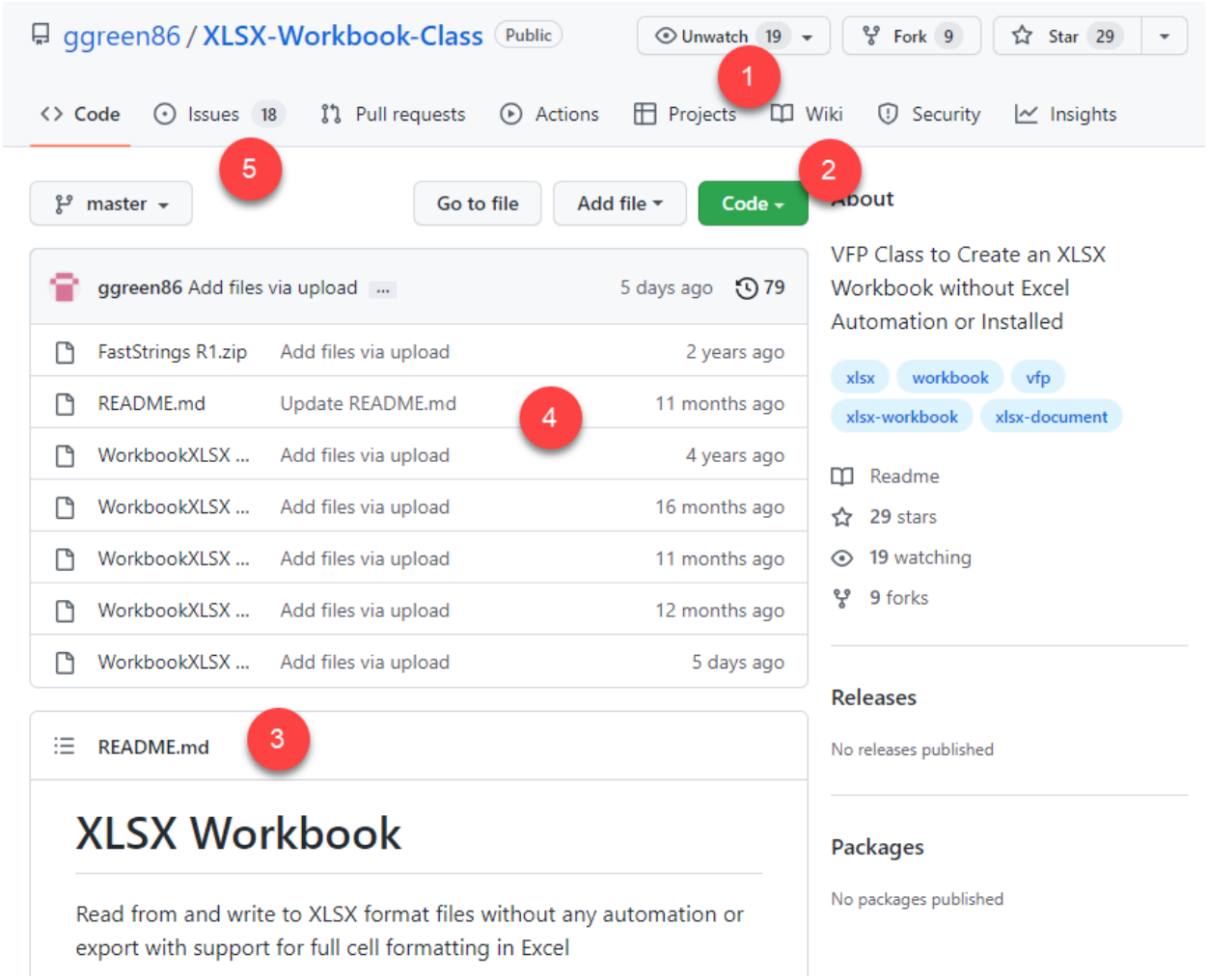


Figure 3. A typical VFPX project repository.

- You can get notification via email when the project is updated by clicking the Watch button (#1 in Figure 3) and choosing the desired option. Note you must be logged in with a GitHub account to turn on watching.
- To install the project on your system, you can either download the project as a zip file or clone the repository; the latter requires Git be installed on your machine (#2 in Figure 3).
- The project's README.md file is a Markdown file (Markdown is similar to HTML but with simplified syntax) GitHub automatically renders on the page (#3 in Figure 3) so it acts as the welcome page and in many cases, provides the documentation for the project as well.
- The source list (#4 in Figure 3) shows the folders and files making up the project. Some projects, such as this one, are just a zip file of source. Most, however, are individual files, such as PRGs, SCXs, and so on. Click a file to view a page containing its content (most binary files can't be viewed, but some, such as PDFs can) and its version history.

- To report a bug, ask a question, or make a feature request, click the Issues tab (#5 in Figure 3) and create an issue. This is much preferred over sending an email to the project manager or posting something on a forum because it allows other users to see and comment on the issue as well.

If you're used to posting issues or comments on the Thor or GoFish Google Groups forums, be aware those forums are being deprecated. The preferred locations going forward are the Issue tab for the Thor and GoFish GitHub repositories.

Git

If you want to clone a project rather than just downloading it, you need to install Git. You can get Git from a variety of sources.

- The main source for Git is at https://git-scm.com/downloads. Choose the Windows version for VFPX projects.
- Git is basically a command-line utility, so there are numerous visual interfaces available, including TortoiseGit (https://tortoisegit.org) and SourceTree (https://www.sourcetreeapp.com). Some of these automatically install Git if it isn't already installed, while others require you install Git yourself.
- Some people like to use GitHub Desktop (https://desktop.github.com), which is a visual interface for GitHub.

To clone a repository, open a command window somewhere and type:

```
git clone RepositoryURL folder
```

where *RepositoryURL* is the URL displayed when you click the Code button in the repository and *folder* is the folder into which to clone the repository (the folder doesn't have to exist). Alternatively, you can use a visual tool to clone the repository. For example, if you use TortoiseGit, you can simply right-click some folder in File Explorer, choose Git Clone, and enter the URL and folder in the dialog that appears.

As I mentioned earlier, I'm not going to go into detail on how to use Git, as there are a lot of resources available to learn Git.

Thor Check for Updates

Another way to install a project is by installing Thor and then using its Check for Updates (CFU) function.

Not all projects support Thor CFU. It's up to the project manager (or someone else) to implement that. We'll see how that's done later.

Also, some projects use an older mechanism for supporting Thor CFU, so the

version number you see in the Thor CFU dialog may not match the one in the project's repository.

Here's an overview of how Thor CFU works. For details, see https://github.com/VFPX/Thor/blob/master/Docs/Thor_Check_For_Updates.md and http://mattslay.com/how-thor-checks-and-distributes-new-versions-of-tools-in-foxpro/.

- Thor CFU downloads Updates.zip from the ThorUpdater folder of the Thor GitHub repository and unzips it in the Thor\Tools\Updates subdirectory of your Thor install folder.
- Thor CFU then runs the PRGs in that subdirectory to get information about each project, mostly the URL for the file containing version information and the URL for the zip file containing the project files.
- It checks to see if a project is installed, looking in either Thor\Tools\Apps or Thor\Tools\Components, depending on whether the project is a "component" or an "app," and if so, compares the version number in its version file with the one downloaded from the project's repository.
- Thor CFU then displays the dialog shown in **Figure 4**.
- To install or update one or more projects, put a checkmark in the Update column and click Install Updates. Thor CFU downloads the zip file for each project, unzips it in the appropriate folder, and creates a text file containing the version number.

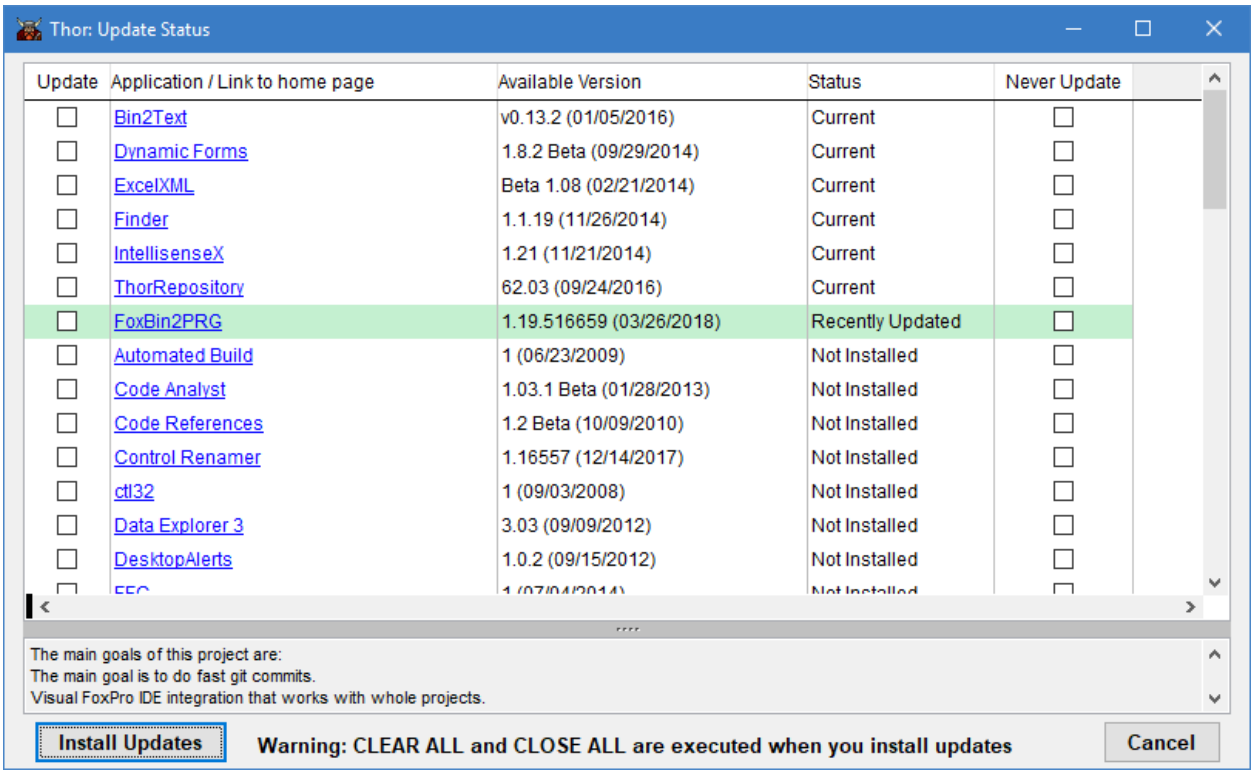
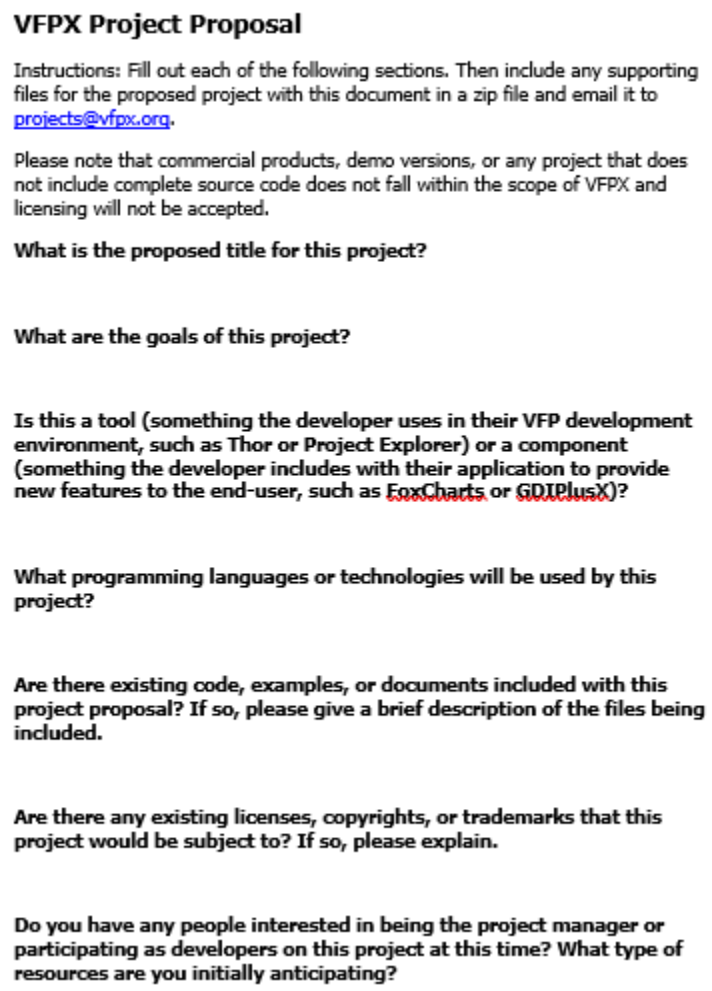


Figure 4. Thor's Check for Updates function can install and update many VFPX projects.

Starting a new project

Suppose you have an idea, or even better, some code, you want to add as a new project to VFPX. To do that, submit a proposal to the VFPX admins. Don't worry: you don't have to write a thesis; just answer a few questions about the project by filling in a Microsoft Word document (**Figure 5**) and email it to the VFPX admins. See the instructions at https://vfpx.github.io/newproject for details.



VFPX Project Proposal

Instructions: Fill out each of the following sections. Then include any supporting files for the proposed project with this document in a zip file and email it to projects@vfpx.org.

Please note that commercial products, demo versions, or any project that does not include complete source code does not fall within the scope of VFPX and licensing will not be accepted.

What is the proposed title for this project?

What are the goals of this project?

Is this a tool (something the developer uses in their VFP development environment, such as Thor or Project Explorer) or a component (something the developer includes with their application to provide new features to the end-user, such as FoxCharts or GDIPlusX)?

What programming languages or technologies will be used by this project?

Are there existing code, examples, or documents included with this project proposal? If so, please give a brief description of the files being included.

Are there any existing licenses, copyrights, or trademarks that this project would be subject to? If so, please explain.

Do you have any people interested in being the project manager or participating as developers on this project at this time? What type of resources are you initially anticipating?

Figure 5. Fill out the VFPX Project Proposal and email it to the VFPX admins.

The VFPX admins will review the project and email you whether it's approved or not. Almost all projects are approved. The main reasons for a project not being approved is it's trivial (such as a small snippet of code) or it competes with a commercial product (such as a web framework competing with West Wind Web Connection). Having your project rejected doesn't mean you can't make it available to the world; you can create your own GitHub repository, post a Dropbox link, or do whatever you wish. It just won't appear in the VFPX project list.

Creating a local repository

The first step (besides creating the source code for the project) is to create a local repository. If you prefer working from the command line, open a command window, CD to the project folder, and type `git init`. Otherwise, use your preferred visual tool. For example, I use TortoiseGit so I select the project folder in File Explorer, right-click, and choose Git Create Repository Here.

The default branch for Git projects was "master" but that's been supplanted by "main" (see https://twitter.com/mislav/status/1270388510684598272 for background). If you wish to use "main" as the default for your projects, open a command window and type:

```
git config --global init.defaultBranch main
```

Once you've created the repository, add some files to it. There are two types of files: those making up the project itself (that is, the source code) and the files related to Git, GitHub, deployment, documentation, and so on.

Project files

The repository obviously must include the source code for the project: PRGs, SCXs, VCXs, PJXs, etc. You should exclude any files VFP generates, such as FXP, MPR, BAK, TBK, and similar files; see the next section for a discussion of using a .gitignore file to automatically specify which files to exclude.

You should create text equivalents of VFP binary files (SCX, VCX, PJX, DBF, etc.). The most populate way is using FoxBin2PRG, a VFPX tool written by Fernando Bozzo and available at https://github.com/fdbozzo/foxbin2prg. Include the text equivalent files (SC2, VC2, PJ2, etc.) in the repository. The benefit of text equivalents is diffing tools such as those coming with Git or a better tool such as Beyond Compare cannot diff binary files but they can diff text equivalents, making it easy to identify changes between commits.

If you use the VFPX Deployment tool discussed in the Deployment section of this document, you don't have to manually create the text equivalents; the tool does that for you as part of the deployment process.

Other files

In addition to the files for the project, other files should be included in the repository:

- README.md is the "home" page for the project. It's automatically displayed when the user navigates their browser to the URL for the project. We'll discuss what content it should have in the Documentation section of this document.
- CONTRIBUTING.md in the .github folder should have clear instructions about how to help with the project. This is discussed in more detail in the Documentation section of this document.

- bug_report.md and feature_request.md in .github\ISSUE_TEMPLATE should be templates for bug reports and feature requests.
- .gitignore should list files to exclude from the repository. Christof Wollenhaupt has some example .gitignore files at https://github.com/cwollenhaupt/foxGitIgnore.

Normally I recommend excluding VFP binary files from the repository, and Christof's .gitignore file specifies that. The reason for that recommendation is opening a binary file in VFP or recompiling the project can change the timestamp on the file even if no changes were made, so it's difficult to tell which files were changed and which were merely touched or recompiled. However, if you exclude binary files, a developer cloning your project has to generate the binary files from the text equivalents, and inexperienced developers may miss that step and become frustrated when they can't find the source code. So, I suggest including VFP binaries in VFPX projects. The sample files accompanying this document includes a modified version of Christof's .gitignore that does not exclude binary files.

- .gitattributes should specify to not alter line endings (that is, adjust line feeds to carriage return/line feeds or vice versa). You can configure Git to not do that but other developers may not have done that configuration, so it's better to specify it at the project level with a .gitattributes file.
- ChangeLog.md contains release notes for the project. We'll discuss this in more detail in the Documentation section of this document.
- We'll discuss project documentation files in the Documentation section of this document.
- We'll discuss Thor Check for Updates files in the Deployment section of this document.

The sample files accompanying this document includes a starter set of files you can use for a new project.

Once all the files are in place, add them to the repository and commit. Using the command line:

```
git add .
git commit -m "message"
```

Using TortoiseGit, I right-click the folder in File Explorer, choose Git Commit -> "main", click the All button to select all files, enter a commit message (the convention is "Initial commit" for the first commit), and click Commit.

Creating a GitHub repository

One of the questions you have to answer in the project proposal is the repository location: should it belong under VFPX (that is, https://github.com/VFPX/*ProjectName*) or your own

account (that is, https://github.com/*YourName*/*ProjectName*)? Although initially all projects were under VFPX, the trend of late has been that new projects are under the developer's account. There are pros and cons to both approaches:

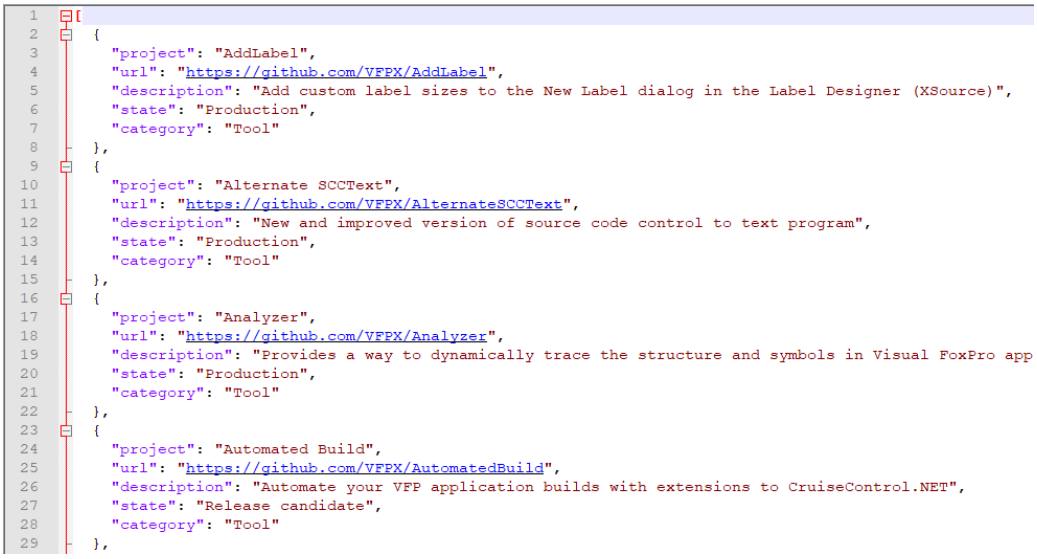
- The most obvious issue with the project being under VFPX is it's not your repository. VFPX admins can add you as a collaborator but ultimately they have control over it.
- If the project's repository is under the developer's account and the developer is no longer available, it's more difficult for others to collaborate on the project. They can fork the repository and issue pull requests (more about that later) but it's possible no one will merge them. Others can comment on issues, but no one may close them. An example: Matt Slay sadly passed away and the repository for his GoFish and Dynamic Forms projects are essentially locked. So, VFPX admins forked them and changed the VFPX project list links to point to https://github.com/VFPX/GoFish and https://github.com/VFPX/DynamicForms, respectively, rather than Matt's repositories.

If you decide the repository belongs under VFPX, a VFPX admin will create the repository and add you as a collaborator. If it belongs under your account, create the repository on GitHub:

- Navigate your browser to github.com and log into your account.
- On the Repositories tab, click New.
- Enter the repository name and description, make sure Public is selected, and click Create Repository.
- Email the VFPX admins the URL for the repository.

In either case, the VFPX admins will add the project to the project list and create a short post describing it.

https://vfpx.github.io, the GitHub Pages site for VFPX, is itself a GitHub repository. The project list is data driven: it uses two JSON files in the _data folder named projects.json (for VFPX projects) and nonvfpx.json (for other projects) as the source of the items in the list. So, the VFPX admins simply add a new record to projects.json (**Figure 6**), create a new Markdown file describing the new project in the _posts folder, and commit and push to the repository.



```
[
  {
    "project": "AddLabel",
    "url": "https://github.com/VFPX/AddLabel",
    "description": "Add custom label sizes to the New Label dialog in the Label Designer (XSource)",
    "state": "Production",
    "category": "Tool"
  },
  {
    "project": "Alternate SCCText",
    "url": "https://github.com/VFPX/AlternateSCCText",
    "description": "New and improved version of source code control to text program",
    "state": "Production",
    "category": "Tool"
  },
  {
    "project": "Analyzer",
    "url": "https://github.com/VFPX/Analyzer",
    "description": "Provides a way to dynamically trace the structure and symbols in Visual FoxPro app
    "state": "Production",
    "category": "Tool"
  },
  {
    "project": "Automated Build",
    "url": "https://github.com/VFPX/AutomatedBuild",
    "description": "Automate your VFP application builds with extensions to CruiseControl.NET",
    "state": "Release candidate",
    "category": "Tool"
  },
```

Figure 6. projects.json contains the list of VFPX projects.

The next step is to push from your local repository to GitHub. You can use the command line if you wish:

```
git push -v --progress RepositoryURL main
```

Since I use TortoiseGit, I right-click the folder in File Explorer, choose Git Sync, click the Manage button, enter the URL for the GitHub repository, click OK (this only needs to be done the first time), and click Push.

Versioning

Versioning a project comes down to one thing: how does the user know how the version of a project they have installed compares to the one on GitHub?

It seems like there are as many ways to provide versioning information for a project as there are projects. Here are some ideas:

- Put the version number in a constant in an include file and use that constant where you want the version number displayed. For example, PEM Editor uses PEMEditorVersion.h, which has something like this:

  ```
  #define ccPEMEVERSION [PEM Editor - 7.41.08 - February 16, 2023 - 20230216]
  ```

  ccPEMEVERSION is used in various places throughout the application to display the version number in forms. When a new version is released, the person making changes updates PEMEditorVersion.h and rebuilds the project.

- Put the version number into a property of a class. The SFMenu class in the OOP Menu project has a cVersion property containing the current version number, although it isn't displayed anywhere.

- Use a major version number that changes with major releases and a minor version number that changes with builds. I like to use the current date minus January 1, 2000 because that gives a number I can use to reverse engineer the release date if necessary. For example, if I release a new build of version 1.0 of a project on February 19, 2023, PADL(DATE() - DATE(2000, 1, 1), 5, '0') gives "08450" as the minor version number, so the version number is 1.0.08450.

One thing to avoid is duplication: having a version number in code (include file, property, etc.) and a separate one in the files used by Thor CFU, because it's likely they'll get out of sync and then it's hard to tell which is correct. We'll look at a mechanism to handle that when we discuss deployment.

Documentation

Ken Levy was famous for writing "self-documenting" code. However, he's a genius. The rest of us need documentation to properly use a project.

The first question to consider is what to document. As a minimum, the documentation should include:

- The name of the project manager.
- The purpose of the project.
- The release date and version number of the latest release.
- Release history notes: release date, version number, and a list of the changes.
- Instructions on installation, use, and deployment (if applicable).
- Instructions on how to help with the project, especially how to release a new version.

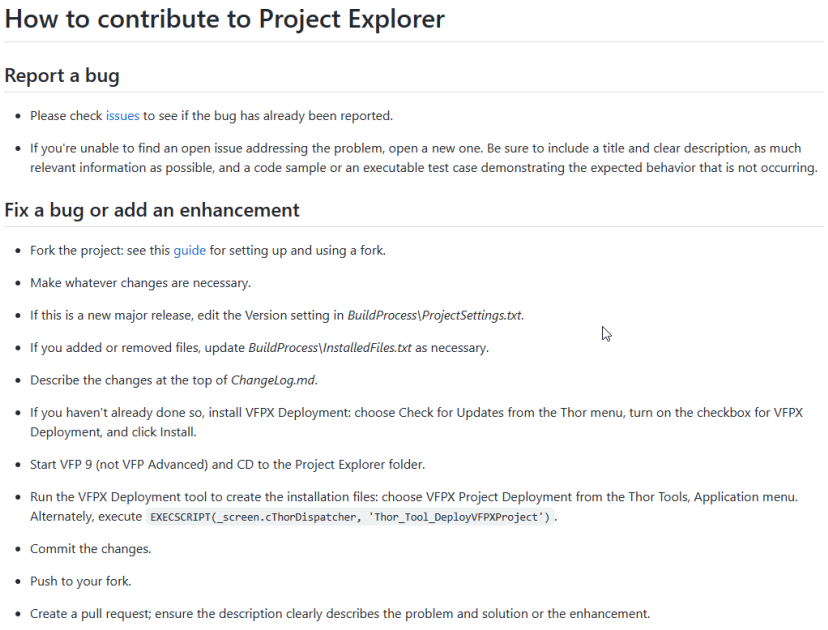
Documenting the internals of the project can provide useful information to other developers who may work on the project (or even yourself a few years down the road).

The next question is the format of the documentation. Most GitHub projects use Markdown files in the repository, but other choices are available:

- GitHub Pages: see https://github.blog/2016-08-22-publish-your-project-documentation-with-github-pages for details.
- Repository Wiki: see https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis and https://gist.github.com/subfuzion/0d3f19c4f780a7d75ba2 for information on using the Wiki included with the repository for documentation.
- Microsoft Word, PDF, and CHM: binary formats aren't good choices, although I admit my Project Explorer project uses a PDF file (generated from Word) because I haven't gotten around to converting it to Markdown.

Assuming you're using Markdown files, the next question is where the documentation goes. Some simple projects, such as Object Inspector, have all the documentation in README.md. However, here's a good design for documentation:

- README.md contains the name of the project, the project manager, a description of the project (perhaps with screen shots to entice the user to investigate further), the release date and version number of the latest release, and links to other documents containing documentation, release notes, and how to help with the project. Note GitHub runs on a Linux server, meaning it's case-sensitive, so be sure to use the correct case for all URLs and use a forward slash (/) rather than a backslash (\) for paths.
- Images go in the images folder.
- ChangeLog.md contains release notes. It should include the project name and an abbreviated project description (perhaps just a single sentence) because if your project supports Thor CFU (discussed later), this appears in the Thor CFU dialog as the project description. This file should go in the project folder rather than a subdirectory (such as docs) so the VFPX Deployment tool (also discussed later) can find it.
- Documentation Markdown files go in the docs folder. If there's more than one document, the one linked to by README.md should be an overview and table of contents, with links to the other documents.
- Documentation on how to make and deploy changes to the project go in CONTRIBUTING.md in the .github subdirectory. **Figure 7** shows this page for the Project Explorer project (https://github.com/DougHennig/ProjectExplorer). Note it gives detailed steps about how to deploy changes made to the project.



How to contribute to Project Explorer

Report a bug

- Please check issues to see if the bug has already been reported.
- If you're unable to find an open issue addressing the problem, open a new one. Be sure to include a title and clear description, as much relevant information as possible, and a code sample or an executable test case demonstrating the expected behavior that is not occurring.

Fix a bug or add an enhancement

- Fork the project: see this guide for setting up and using a fork.
- Make whatever changes are necessary.
- If this is a new major release, edit the Version setting in *BuildProcess\ProjectSettings.txt*.
- If you added or removed files, update *BuildProcess\InstalledFiles.txt* as necessary.
- Describe the changes at the top of *ChangeLog.md*.
- If you haven't already done so, install VFPX Deployment: choose Check for Updates from the Thor menu, turn on the checkbox for VFPX Deployment, and click Install.
- Start VFP 9 (not VFP Advanced) and CD to the Project Explorer folder.
- Run the VFPX Deployment tool to create the installation files: choose VFPX Project Deployment from the Thor Tools, Application menu. Alternately, execute `EXECSCRIPT(_screen.cThorDispatcher, 'Thor_Tool_DeployVFPXProject')`.
- Commit the changes.
- Push to your fork.
- Create a pull request; ensure the description clearly describes the problem and solution or the enhancement.

Figure 7. CONTRIBUTING.md should have instructions about how to help with the project.

Deployment

If you don't want to support Thor CFU, deployment to GitHub is straightforward:

- Use FoxBin2PRG to create or update the text equivalents of VFP binary files.
- Commit your changes.
- Push to the remote repository.

However, supporting Thor CFU is a good choice because it not only makes the project initially more visible (it automatically appears in the Thor CFU dialog), it also makes notifying users when a new version is released simple and installing and updating an easy operation for the user.

The best way to support Thor CFU is to use the VFPX Deployment tool. This tool itself supports Thor CFU, so installing it is as simple as choosing Check for Updates from the Thor menu, turn on the checkbox for VFPX Deployment, and clicking Install Updates. This adds a "VFPX Project Deployment" item to the Thor Tools, Applications menu.

The documentation in the project repository (https://github.com/VFPX/VFPXDeployment) has details on how to use it, but here's an overview:

- The BuildProcess folder contains files specifying settings for the project, such as current version number, and how VFPX Deployment should deploy it.
- The InstalledFiles folder is a staging area: VFPX Deployment zips the files in this folder into *ProjectName*.zip in the ThorUpdater folder. This is the file Thor CFU downloads and unzips.
- The ThorUpdater folder contains *ProjectName*.zip and *ProjectName*Version.txt, the version files Thor CFU downloads to see if there's a new version available.

BuildProcess contains the following:

- ProjectSettings.txt contains the project settings.
- VersionTemplate.txt is a template file used to generate ProjectNameVersion.txt.
- InstalledFiles.txt is an optional file listing which files to copy to the InstalledFiles folder. If it doesn't exist, you have to populate the InstalledFiles folder yourself. Some projects, such as Object Explorer, use InstalledFiles as the source folder, so there's no need to copy anything.
- BuildMe.prg is an optional program containing additional code to customize the deployment process.

One thing to keep in mind is to differentiate between what's in repository (that is, needed for someone working on the project) and what's deployed with CFU (that is, what's needed to use the tool). For example, you don't have to install README.md or any other Markdown files, documentation, text equivalents, etc. If the project is delivered as an APP or EXE,

source code isn't needed either. If someone wants to look at the source, read the documentation, or make changes to the project, they can clone the repository or download the source. That being said, there's no harm in installing these files and many projects do.

Let's look at an example. **Listing 1** shows the content of ProjectSettings.txt for Project Explorer.

Listing 1. ProjectSettings.txt for Project Explorer.

```
AppName   = Project Explorer
AppID     = ProjectExplorer
Version   = 1.0
Component = No
ChangeLog = ChangeLog.md
PJXFile   = projectexplorer.pjx
```

These settings tell VFPX Deployment to create files named ProjectExplorer.zip and ProjectExplorerVersion.txt in ThorUpdater; the current version is 1.0; Thor CFU should install this in the ProjectExplorer subdirectory of Thor\Tools\Apps because Component is "No;" the content of ChangeLog.md should be included in ProjectExplorerVersion.txt; and it generates text equivalents of the VFP binaries included in ProjectExplorer.pjx and rebuilds ProjectExplorer.app from ProjectExplorer.pjx.

Listing 2 shows the content of InstalledFiles.txt. VFPX Deployment copies these files to the InstalledFiles folder, preserving the folder structure. Notice source code is not included, only the necessary files to run Project Explorer.

Listing 2. InstalledFiles.txt for Project Explorer.

```
ProjectExplorer.app
ProjectExplorerSettings.xml
system.app
Addins\*.prg
Addins\template.txt
Addins\folder.png
Addins\Functions\*.prg
```

I won't show the content of VersionTemplate.txt because it's created the first time VFPX Deployment is used and I didn't make any changes to it. There's no BuildMe.prg.

When I want to deploy a new version of Project Explorer, I do the following:

- Make whatever changes are necessary and test.
- Change the version number in BuildProcess\ProjectSettings.txt.
- Edit ChangeLog.md to describe the changes.
- Choose VFPX Deployment from the Thor Tools, Applications menu.

After a moment, ProjectExplorer.zip and ProjectExplorerVersion.txt in ThorUpdater have been updated, so I commit and push the changes to the repository. The next time someone chooses Check for Updates from the Thor menu, they'll see a new version of Project Explorer is available.

Even if you use VFP Advanced, you must build APP files for VFPX projects using VFP 9 because while VFP Advanced can run APP files created in VFP 9, the opposite isn't true, so users not using VFP Advanced wouldn't be able to use your APP file if you built it using VFP Advanced. VFPX Deployment enforces this: if ProjectSettings.txt specifies building an APP or EXE file, it gives a warning and terminates if you're running VFP Advanced.

Testing Thor CFU

For a new project, there are two more, one-time steps: testing and adding to Thor CFU.

You should test your deployment on your own machine by doing the following:

- In the Thor\Tools\Updates subdirectory of your Thor install folder, create a My Updates folder if it doesn't already exist.
- Copy the Thor updater file VFPX Deployment generated, named Thor_Update_*ProjectName*.prg in the BuildProcess folder of your project, to the My Updates folder.
- Choose Check for Updates from the Thor menu. As **Figure 8** shows, your project appears in italics indicating it's a "my update" update. ("Do Nothing" is a test project I created.)

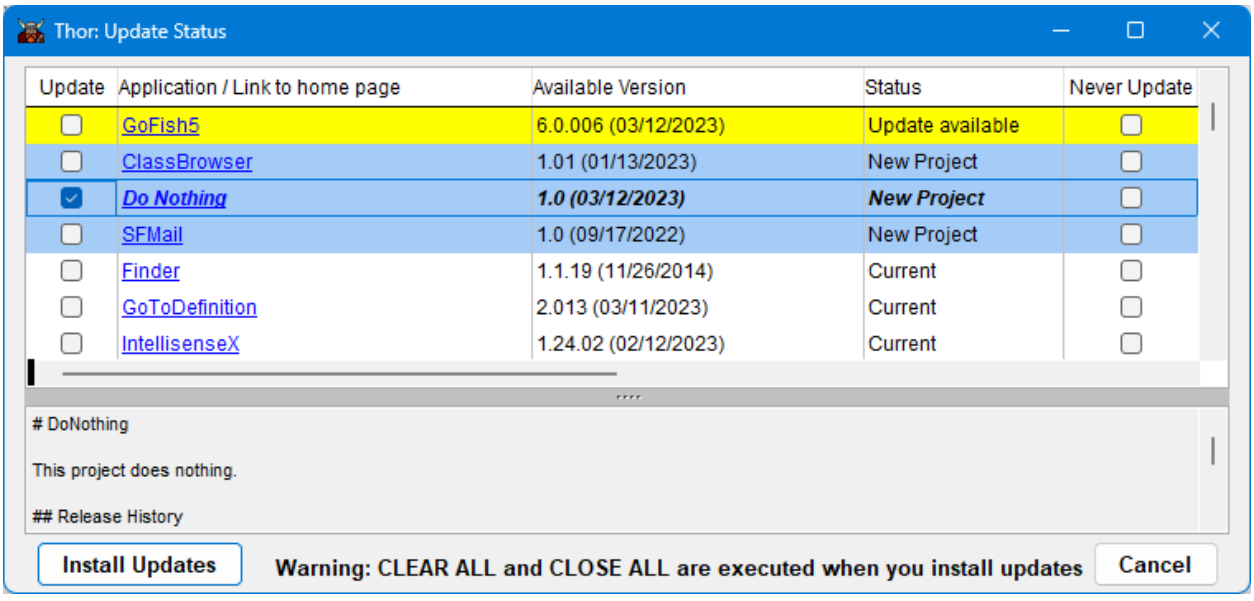


Figure 8. Your project appears in italics in the Thor CFU dialog.

Install the project and check the Thor\Tools\Components or Thor\Tools\Apps subdirectory of your Thor install folder (depending on whether you set Component = Yes or No in ProjectSettings.txt). Make sure all the necessary files are there and the version file, named *ProjectName*VersionFile.txt (specified in Thor_Update_*ProjectName*.prg), exists and contains the correct version information. If you need to change anything, delete the project folder under Thor\Tools\Components or Thor\Tools\Apps, make the necessary changes, commit and push, and try Thor CFU again.

Once you've confirmed the Thor CFU process works, send Thor_Update_*ProjectName*.prg to the VFPX admins at projects@vfpx.org (zip it or change the extension to TXT since Microsoft Outlook blocks PRG attachments). They'll add it to Thor (specifically, it's included in Updates.zip in the ThorUpdater folder of the Thor GitHub repository, as well as in the Updaters\Updates folder for diffing purposes) so it's available to other VFP developers through Thor CFU.

Versioning

One thing to avoid is having the version number in more than one place. For example, the SFMenu class in the OOP Menu project has a cVersion property. When I release a new version, I change both that property and the version number in ProjectSettings.txt. If I update one and forget to do the other, the version numbers are out of sync.

PEM Editor has a better approach. It uses a BuildMe.prg (**Listing 3**) for two reasons: since a reference to it may be in memory, we need to release that before trying to build PEMEditor.app, and we want to automate updating the version number so it isn't a forgotten step.

Listing 3. BuildMe.prg for PEM Editor automates updating the version number.

```
* Release references to PEMEditor.app since we'll be rebuilding it.

if type('_oPEMEditor') = 'O'
   _oPEMEditor.Release()
endif type('_oPEMEditor') = 'O'
release _oPEMEditor

* Update the version information in PEMEditorVersion.h.

text to lcVersion noshow textmerge
#define   ccPEMEVERSION [PEM Editor - <<pcVersion>> - <<cmonth(pdVersionDate)>>
<<day(pdVersionDate)>>, <<year(pdVersionDate)>> - <<strtran(pcVersionDate, '-')>>]
endtext
strtofile(lcVersion, 'downloads\source\pemeditorversion.h')
```

As discussed earlier, PEMEditorVersion.h is one of the include files for PEM Editor source code, so when VFPX Deployment builds the app, the updated version number is used. pcVersion is a public variable containing the version number specified in ProjectSettings.txt, pdVersionDate is a public variable containing the release date (today's date if it isn't specified), and pcVersionDate is the text equivalent of pdVersionDate.

Contributing to a project

By "contributing," I don't mean financially but helping by writing or updating documentation, fixing bugs, or adding enhancements. As discussed earlier, the project should have detailed notes on how to contribute in CONTRIBUTING.md, which should be linked to in README.md.

The way you contribute to a project is to fork the project to your own repository, make whatever changes are necessary, and then creating a pull request. The project manager reviews the pull request, may start a discussion about the changes with you, and once satisfied, may merge the pull request into the repository. Let's look at these steps.

Forking a project

To fork a project, click the Fork button on the project's repository (**Figure 9**) to create a new repository under your account. The default name for the new repository is the same as the original but you can change it if you wish. After forking the project, clone its repository to your machine.

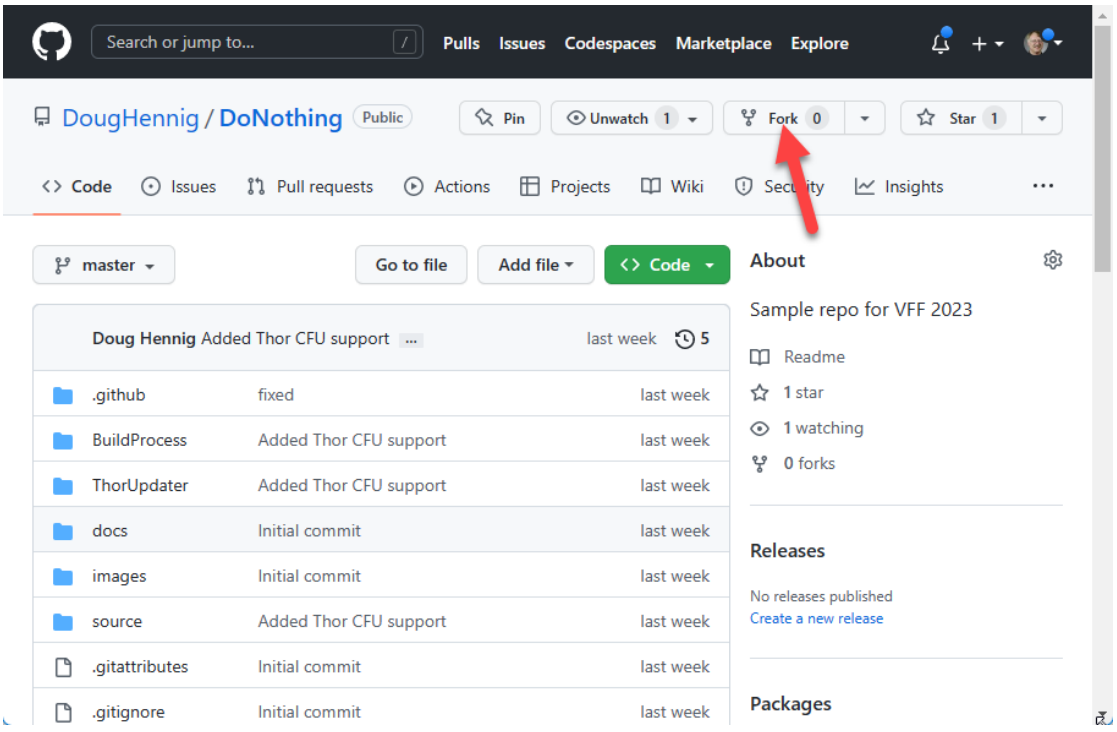


Figure 9. Click the Fork button to fork a repository.

Making changes

Make the necessary changes to the project. (I'm not going to discuss creating and working on a different branch here). Don't forget to follow the instructions in CONTRIBUTING.md for building, updating documentation, versioning, using VFPX Deployment if necessary, etc.

When the changes are done and tested, commit your changes and push to the fork repository.

Creating a pull request

A pull request is a request to pull from a fork repository to the original repository and merge the changes into the original. Navigate your browser to the fork repository, choose the Pull Requests tab, and click New Pull Request (**Figure 10**). You can enter any comments you wish about the pull request, then click Create Pull Request. You'll notice the browser navigates to the original repository; this confused me the first time because the repository name was the same and I didn't notice the account was different.

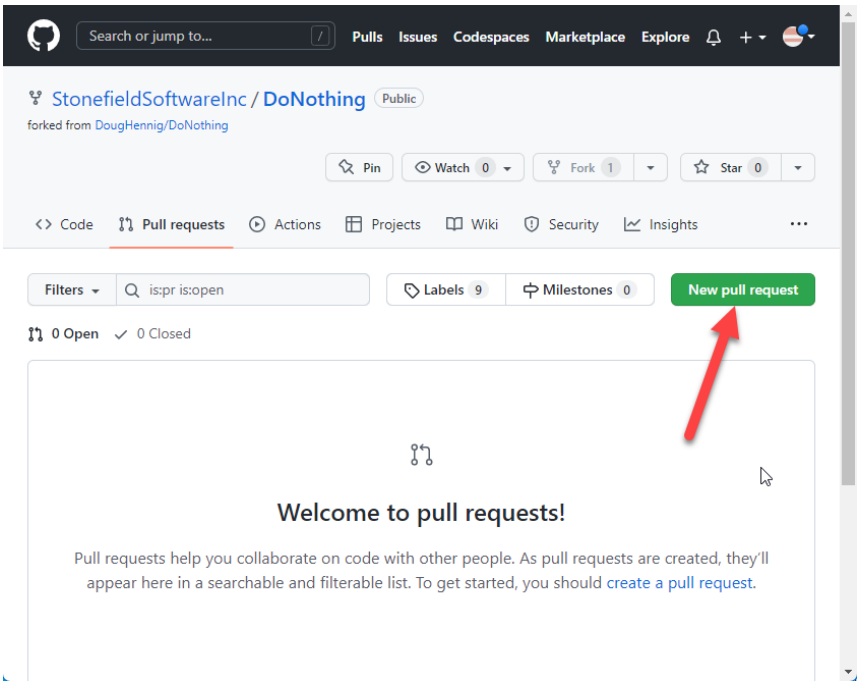


Figure 10. Click the New Pull Request button to create a pull request.

The admin for the original repository gets an email notifying them about the pull request. They may start a conversion on the pull request, asking for more information or a discussion about the changes, to which you can reply. There may be conflicts between your changes and changes someone else made, so those conflicts must be resolved before they can be merged.

Once the admins is ready, they can merge the changes into the repository by clicking Merge Pull Request (**Figure 11**). After that, you can continue to make changes in your fork and issue pull requests, or if you think that's the only change you'll make, you can delete your fork repository. If you change your mind, you can recreate a fork.

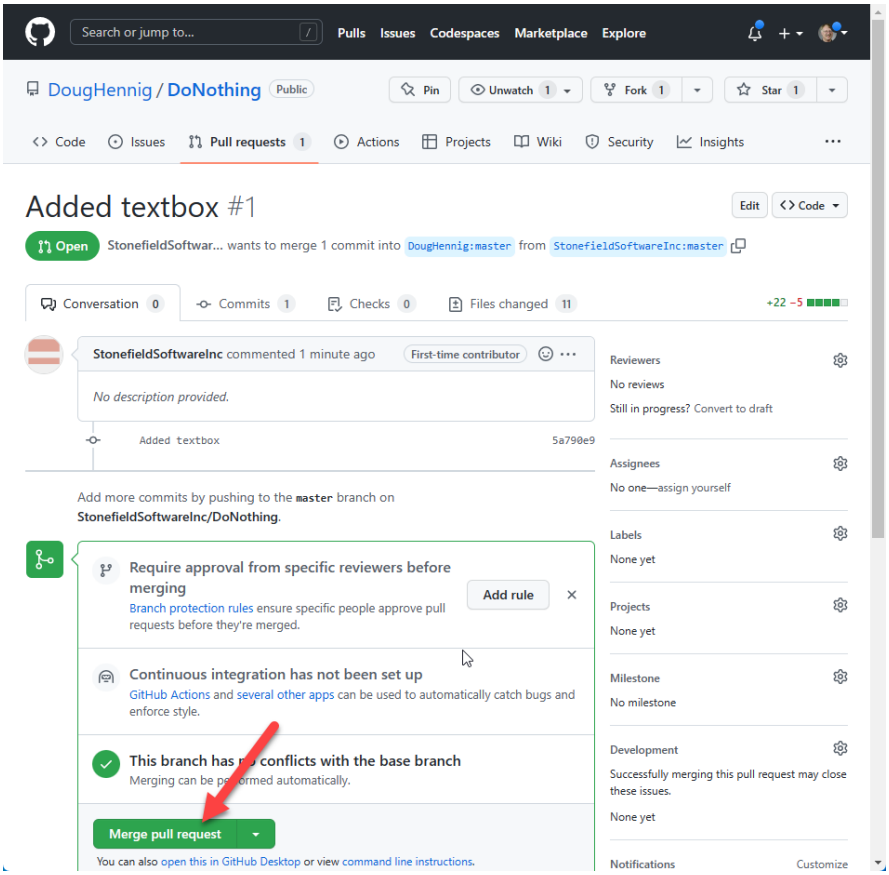


Figure 11. The repository admin merges the changes from the fork by clicking Merge Pull Request.

Resources

If you're interested in reading more about certain VFPX projects:

- The Technical Papers page of my personal web site has articles doing deep dives on several VFPX projects, including Themed Controls, Dynamic Forms, and FoxCharts.
- The Articles page of Tamar Granor's web site has several in-depth articles on Thor, Object Inspector, and FastXTab.
- *VFPX: Open Source Treasure for the VFP Developer* is a little out of date but still a great resource for many VFPX projects, especially its section on Thor.

There's a ton of resources available for Git and GitHub. Some I've used are:

- GitHub documentation: https://docs.github.com.
- Fernando Bozzo, who has contributed numerous VFPX projects, has some excellent Git information at https://github.com/fdbozzo/git_training.
- Scott Hanselman's blog post "The Squishy Side of Open Source" has some good ideas for those new to open source.

Summary

I hope this document inspired you to contribute to VFPX, either by proposing your own project or contributing to an existing one. Even if that's not your thing, hopefully you can use the ideas in this document with your own repositories.

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.