Chapter 1
Introduction

Building an application framework is a challenging and rewarding experience. This book will teach you how to build your own framework by using examples while offering guidance.

Who should read this book?

This book provides some perspective on how to build a polished and robust application. It is intended for intermediate and experienced programmers who want to learn to create their own framework or who are interested in learning techniques to enhance their own work. If you are comfortable with programming, regardless of the language, you should be able to follow along.

If you are new to programming, the Visual FoxPro Help file is an excellent reference while you are learning the syntax and mechanics of programming. Additionally, there are many excellent books in the Hentzenwerke collection to guide you in the early stages of learning application development. Take your time. Refer to these resources as you read about the concepts presented in this book. Fully understanding these concepts will make you a much stronger programmer, as opposed to "going it alone."

What is an application framework?

A framework is a reusable integration of components engineered to facilitate development of a particular type of application.

This definition of a framework is based on the definition provided in Design Patterns *(Gamma, Helm, Johnson, Vlissides; Addison-Wesley, 1995). The definition supplied in the book: "A framework is a set of cooperating classes that make up a reusable design for a specific class of software."*

The representation of a framework varies with the type of application being built. The following example illustrates what constitutes a component, an application, and a framework.

Consider an automated teller machine (ATM). The software you interact with is an example of a client/server application. Each part of the application—the bankcard reader, the screen display, the buttons on the keypad—is a component of the application. Next, consider how the user interacts with an ATM while withdrawing money from his or her account. The sequence may be summarized as follows:

1. Swipe your card.
2. Type in your password.
3. Select an account (Checking, Savings…).
4. Enter the amount you want to withdraw.

5. Confirm that you want to withdraw the money.
6. Retrieve the money from the machine.
7. Indicate that you do not want to conduct another transaction.

Consider step 5: "Confirm that you want to withdraw the money." The event sequence that occurs when the user confirms the withdrawal might be as follows:

1. Make a connection to the back-end database server.
    - Did the connection succeed?
        - Continue
    - Did the connection fail?
        - Inform the user that the system is unavailable at this time.
2. Does the ATM have enough cash to complete the transaction?
    - Yes
        - Continue
    - No
        - Inform the user that the machine does not have sufficient funds to complete the transaction.
3. Verify the transaction.
    - Does the user have enough money in his/her account to cover the transaction?
        - Yes
            - Continue
        - No
            - Inform the user that he/she does not have sufficient funds to complete the transaction.
4. Post the transaction.
    - Reduce the user's account by the amount of the withdrawal.
    - Reduce the machine's cash balance by the amount of the withdrawal.

An alternative to having the application developer program all facets of the transaction is to use a framework. In this example, the framework developer would be responsible for coordinating the major elements of the transaction. Additionally, the framework developer would most likely take a holistic perspective and define the problem differently.

1. Data connection:

    - At any time, if a connection fails, show a "System Not Available" message.

- Periodically check to see if a connection can be re-established.
- If a connection can be re-established, show the "Welcome" screen.

2. For all transactions:

- Assume the connection to the database is valid.
- Provide a place for the developer to validate the information.
- Run the validation code.

If the validation code fails:

- Show the message generated by the application developer indicating the information supplied is not valid.

If the validation code succeeds:

- Proceed with the posting process.
- Provide a place for the developer to post the transaction.
- Run the post code.

If the transaction fails:

- Show the message generated by the application developer indicating the transaction failed.

If the transaction succeeds:

- Show the "Transaction Succeeded" message.

So what's left for the application developer? Developing the validation and transaction code (business rules) and appropriate message screens (interface design).

This example illustrates the roles of component, framework, and application developers. As you can see, the component developer is associated with the functionality of a single task—for example, connecting to a database. The framework developer focuses on coordinating existing components and adding general functionality, while the application developer provides application-specific functionality.

Why use a framework?

Developing polished, robust applications is a complex and time-consuming task. A framework handles many of the repetitive tasks associated with application development, allowing the application developer to focus on the specific features unique to a specific application.

Frameworks offer major enhancements to productivity. As demonstrated in the illustration, much of the integration code is written and tested prior to starting the application. True, there is a learning curve associated with using a framework. However, when you are familiar with the

framework and the methodologies it employs, you will find that your focus shifts more toward the tasks of the application rather than the details of the implementation.

Varying skill sets are required to develop an application; a few of these are user interface development, database design, and writing business logic. Frameworks can be used to segregate these tasks, allowing you to focus on one area of development at a time. This is particularly helpful in a team environment where certain developers may be more adept in one area of development.

Frameworks tend to solve problems by using a standardized approach. Approaching each development effort in a consistent manner makes transitioning between applications, between developers, or even between different parts of your own code more efficient and less prone to error. Reusable designs that follow a standardized approach can be automated. Frameworks generally provide wizards and builders that reinforce the framework methodology. Wizards and builders further improve productivity by reducing the effort to implement an application.

How is a framework different from a class library?

A class library is a collection of programming tools or components a developer may use when building an application. Components in a class library accomplish specific, often complex, tasks. To use a component, you do not need to know the details of how the task is completed. For example, a hammer is a carpenter's tool. It can drive a nail or remove a nail. To use the hammer, the carpenter does not need to know about metallurgy, physics, or ergonomic design.

Components offer discrete functionality useful in a variety of applications. The data connection discussed at the beginning of the chapter is an example of functionality that could be made into a component. The developer of this component would be knowledgeable about database connection protocols and the types of issues that cause connections to fail. Framework and application developers benefit from using the connection, but do not need to know "how" it works. Components are useful to both application and framework developers.

The framework developer defines the relationship between components and provides much of the code that integrates them. The framework developer leaves "empty spaces" for the application developer to fill in later, defining only the specifications for the desired functionality. When deciding how and when to integrate components, a framework developer follows a plan called the "framework architecture," which is often formalized in an architecture document.

A framework developer typically organizes basic framework functionality into a class library, but this is not required.

The type of application the framework supports dictates the components that a framework contains and how they are integrated. For example, the requirements and structure of an Internet application are entirely different from a traditional FoxPro (fat client) application. Internet applications are stateless, requiring a host of services to manage information between requests. State-full applications "remember" information and require none of the state-management functionality.

Elements of a framework

A framework is comprised of a variety of tools, services and modules. The tools and services provided by each framework will vary depending upon the goals of the framework and type of application to be developed. Here is a listing of the key framework elements:

- Procedures
- Classes
- Tools and components
- Application services
    - Messaging
    - Event logging
- Forms management
- Data management
- Business objects
- Interface controls
- System navigation
    - Menus
    - Toolbars
    - Buttons
- Security
- Error handling

Reasons to create your own framework

Building your own framework offers several advantages that a purchased framework simply cannot. Most important is the thorough understanding of exactly what is happening at each moment in your code.

Learning to use someone else's framework takes time. A lot of time. Developing your own framework takes even more time, but you can be certain that the resulting product will meet your requirements.

At some point your framework, regardless of who developed it, will not meet your needs. Commercial frameworks are intended for a wide audience and may contain features or a certain level of complexity that are of no benefit to you. There is no feeling more frustrating, and no task less productive, than trying to modify someone else's framework to accomplish something for which it was not designed.

A custom framework is lean, supporting only your personal methodology and preferences. When you've properly designed a framework, reworking it to meet a new set of requirements takes less time than reworking someone else's framework.

I started building my first class library as a learning opportunity. Each time I learned something new, it went into the library. I then started to focus on how I was building my applications. Sure, they all worked, but was I building them in the best way possible? Unlike

building a class library, where I was cataloging what I learned, building a framework actually made me a stronger, more well rounded, and more productive programmer.

Reasons to avoid developing your own framework

The biggest drawback to building a quality framework is the time required. You may feel as though much of the work required to produce a framework would have been incurred by writing the application. This is partially true. An extra level of effort is required to ensure that the components you create will work beyond the situation for which they were created. For example, it is not enough to know how to save data in a buffered table; you have to know how to save data in *all* tables, without knowing beforehand the names of the tables or their buffering status.

This leads to a second drawback. It's difficult to develop a framework while you are developing a single application. The requirements used to define the needs of the application are generally not reflective of applications in general. Viewed from the narrow perspective of one project, the following requirement might be missed.

```
Requirement: A developer must have the ability to limit the number of instances
of a form available to the end user.
```

The final reason to avoid developing your own framework is that your experience, imagination, or talents are limiting factors. The other side of the learning curve is that you didn't include all the things you didn't know, or didn't think of.

Framework mindset

When you are developing a framework, you are creating a product that other developers will use to help them build applications. Your job as the framework developer is to make the application developer as productive as possible. Even if you are developing the framework for your own use, it is helpful to separate your roles as framework developer and application developer. When you have your framework hat on, you are thinking, "How can I make my next project (application) better?" When you have your application hat on, you are thinking, "How can I best meet the specific needs of my client?"

Throughout the book, the terms "you" and "framework developer" are synonymous. These terms are used when discussing tasks you should be completing or your objectives when designing and implementing your framework. The terms "application developer" or "developer" refer to the developers that will use your framework. The term "client" is used to represent the project stakeholder—the person or group for which the application is written. If you are a corporate developer, your clients may be members of your company in other departments. If you are a consultant, clients are the ones writing the checks. The term "end user" or "user" represents people that use an application built with your framework. An end user may or may not be a client.

These terms are used throughout the book to clearly distinguish between developing a framework and developing an application from the framework you've created.

Overview

Building a framework can be broken into four phases: planning, component development, integration, and productivity enhancement.

During the planning phase you define requirements, select an architecture, and create a project plan. These activities and the concepts required to complete these steps successfully are covered in the first three chapters of the book.

The component-development phase represents the start of the coding phase. In this phase you develop classes that are responsible for one thing. This one thing could be simple, such as a class that knows how to format and display ZIP code information, to a complex task such as starting an application. Another aspect of developing components is creating the base components that integrate the architectural elements of your framework. Chapters 3 through 9 focus on base components, while chapters 10 through 13 focus on major architectural components.

The integration phase involves linking functionalities that exist in two or more classes. In some cases, you will link classes together directly to create framework components. In other cases, you will create subcomponents that are in turn coordinated to create framework components or that are shared among many components. Chapters 14 through 16 illustrate several ways to link the elements of your framework together.

Finally, with the integration complete, you will want to create tools that reinforce your design decisions and that can help to relieve the mundane aspects of development. Chapters 17 through 19 look at some critical tools you will need to support your framework and make your development efforts more productive.

How should I read this book?

Sequentially. Then, once you have read it cover to cover, refer back to specific sections as needed.

Each chapter builds on the chapter that precedes it. Constantly repeating material from previous chapters is redundant and unpleasant to read. Each chapter assumes that you have read the chapters before it.

The purpose of developing a framework is to make developing an application simpler or more efficient. Take the time to review the samples provided. At times, the framework code is complex. The examples illustrate how the framework can simplify development, even though the framework code may be complex.

A word about the approach presented in this book

In this book, I'll lead you through the process of building a framework called "MyFrame." Each step of the development process is explained in detail, citing general, as well as FoxPro-specific, design considerations you will encounter as you build your framework.

There is no "one way" to create a framework. If there were, it would have been created already and we would all be using it. Creating a framework has more to do with coordinating the many components involved in the framework rather than how a particular feature is implemented. Your framework will be as much a reflection of your personal preferences as the types of applications you build.

The entire book is one big example of how to create a framework. In several chapters, I've prepared examples that illustrate how the framework may be used. The final chapter includes a comprehensive example illustrating how to use the framework. If you don't like the way a particular item is implemented, feel free to change it; after all, it is your framework.

In addition, FoxPro is a wonderfully rich programming environment. As a rule, any task that can be implemented with FoxPro can be implemented in more than one way. This book is

about coordination and planning, not exploring the many ways FoxPro avails itself to solving a particular problem.

Your framework will not be developed in one big blast of development nirvana. Developing a framework is an iterative process. Most likely, your first pass will be to implement the features you require. After using the framework for a while, you might want to implement additional features, and possibly change some things established in the prior pass. After a while, the cycle is repeated again … and again … and, well, you get the idea.

As each requirement is fulfilled and each layer added, you will understand the dependencies that exist between components, how to avoid creating dependencies, and the types of considerations you'll face when requirements change.

Unfortunately, this type of development is difficult to capture in a book. As the reader, you expect, and rightly so, to be able to find the chapter titled "Forms." Scattering functionality throughout the book would make this book difficult to use as a reference when developing your own framework. Instead, consider this book as the first iteration, or "Version 1.0" of your framework.

Icons used in this book

Throughout this book, you'll see the following icons used to point out special notes, tips, and download information.

Information of special interest, related topics, or important notes are indicated by this icon.

Tips—marked by this icon—include ideas for shortcuts, alternate ways of accomplishing tasks that can make your life easier or save time, or techniques that aren't immediately obvious.

*Paragraphs marked with this icon indicate that you can download the referenced code or tool from **www.hentzenwerke.com**.*

Summary

This book is about defining framework objectives and choosing implementation strategies that maximize the potential for reuse. Framework design encompasses all aspects of application development: general programming principles, object-oriented programming techniques, requirements analysis, refactoring, user interface development, class interface development, documentation, and a thorough understanding of the problem domain.

Developing a framework requires you to *put it all together*. Upon completing your framework, you will be a better, more productive developer who is able to produce full-featured applications efficiently.

Updates and corrections to this chapter can be found on Hentzenwerke's Web site, **www.hentzenwerke.com**. Click "Catalog" and navigate to the page for this book.