Preface

Many new users come to LabVIEW with some experience in another programming language. They already know about loops, arrays, conditional statements, subroutines, file I/O, etc. Because of this, many expect that LabVIEW will be easy to learn. After all, it's just another programming language: memorize some syntax rules, figure out how loops and conditional statements work, keep a list of commonly used functions handy, and they should be fluent in no time, right?

Yet they find that it is harder than that. Beginning and intermediate LabVIEW programmers often suffer from "blank page" syndrome. They understand the basic elements and structures of LabVIEW, but they have difficulty knowing where or how to start a new project. They often find they easily can produce functional code but struggle to produce maintainable code. The code produced is often difficult to read—even by the original developer. The problem is that most beginning and intermediate LabVIEW programmers simply don't know what a good LabVIEW program looks like. LabVIEW programmers often produce spaghetti code because, while they understand LabVIEW programming elements, they do not understand how to put them together effectively. Since LabVIEW is a graphical programming language, spaghetti code really does look like spaghetti (Figure 0.1). Spaghetti code may be temporarily functional, but woe to the programmer who needs to extend its functionality—even if that programmer is the original author.

LabVIEW programmers often exist in isolation. Usually there is a single LabVIEW programmer in a work group, leading to something that might be called "the lone wolf syndrome." Lone wolves have no LabVIEW frame of reference, no experienced co-workers to review their code, and no contact with the greater LabVIEW community. As long as the lone wolf produces functional code—no matter the quality—he or she is considered to be the local LabVIEW guru.

In many ways, LabVIEW is easier to learn than a text-based language, but few other languages follow the dataflow paradigm. Many programming concepts carry over to LabVIEW directly, but the elements of good LabVIEW style and proven LabVIEW design patterns must be learned from scratch. Programmers must learn to "think" in LabVIEW.

Coaching by an experienced developer is the best way to learn style and design in any computer language. When a mentor is unavailable, the next best way is to learn through books. Unfortunately, the number of LabVIEW books is fairly small. The few general LabVIEW books aimed at beginners tend to describe the basic elements of LabVIEW rather than describe how to integrate those elements into an effective LabVIEW program.

This book is different. It does not describe every LabVIEW function in excruciating detail. In fact, the set of LabVIEW functions used in this book is quite small.

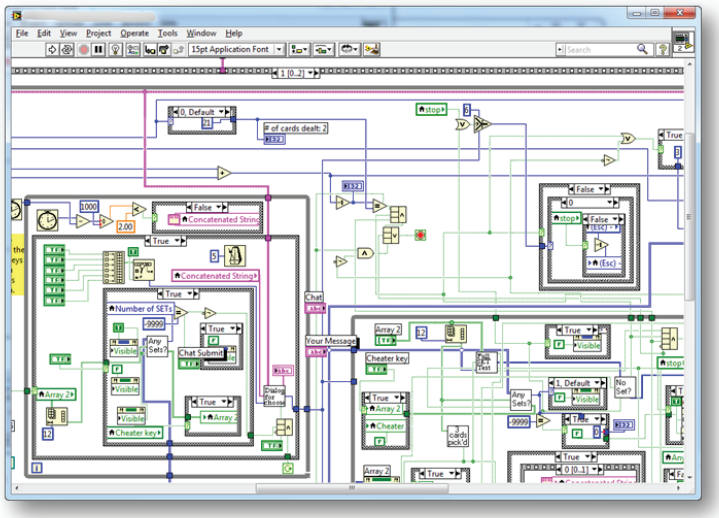

Figure 0.1: Spaghetti code

This book demonstrates what good LabVIEW programs look like. It does this using core LabVIEW functions and common design patterns for a common project drawn from the Certified LabVIEW Developer exam. These patterns build on each other. They provide a firm starting point for most beginning and intermediate projects.

My hope is that new LabVIEW users will absorb these design patterns and implement them in their earliest projects. In this way, they may avoid joining the ranks of LabVIEW spaghetti coders. I also hope that current spaghetti coders will be inspired to re-examine their body of work and re-evaluate their programming practices. It takes humility to admit that your code may not be as good as you once thought, and it takes effort to change your programming habits. But in return, this investment in effort and humility will take your LabVIEW coding to the next level. It did for me.

My final hope for this book is that it will serve as a bridge between the many LabVIEW lone wolves and the greater LabVIEW community. Good programming style and effective design patterns are not developed overnight. They are formed through many years of reflection on the practical experience of the LabVIEW community. By adopting these practices and patterns, the lone wolf can connect with the community, benefit from its experience, and strengthen the community in return.

I would like to thank a number of people who helped me bring this book into being. Andrew Watchorn at National Instruments listened to my ideas for a book that I would like to write "someday" and immediately called NTS Press, setting this book into motion. Thanks to Tom Robbins, my publisher, for shepherding this book with patience. Special thanks to Stephen Mercer, Fabiola De La Cueva, Justin Petry and Jared Kirschner for reviewing my manuscript and vastly improving the final product. Also, I would like to thank Jason Goode, Joe Silva, and David Schlickeisen of NI for producing the cover. Thanks also to Rose Kernan and her crew for putting the book into the final form you see here. Finally, thanks to my wife Heather and my boys, Nathan and Zachary, for allowing me to spend time on this book that would have been better spent with them.