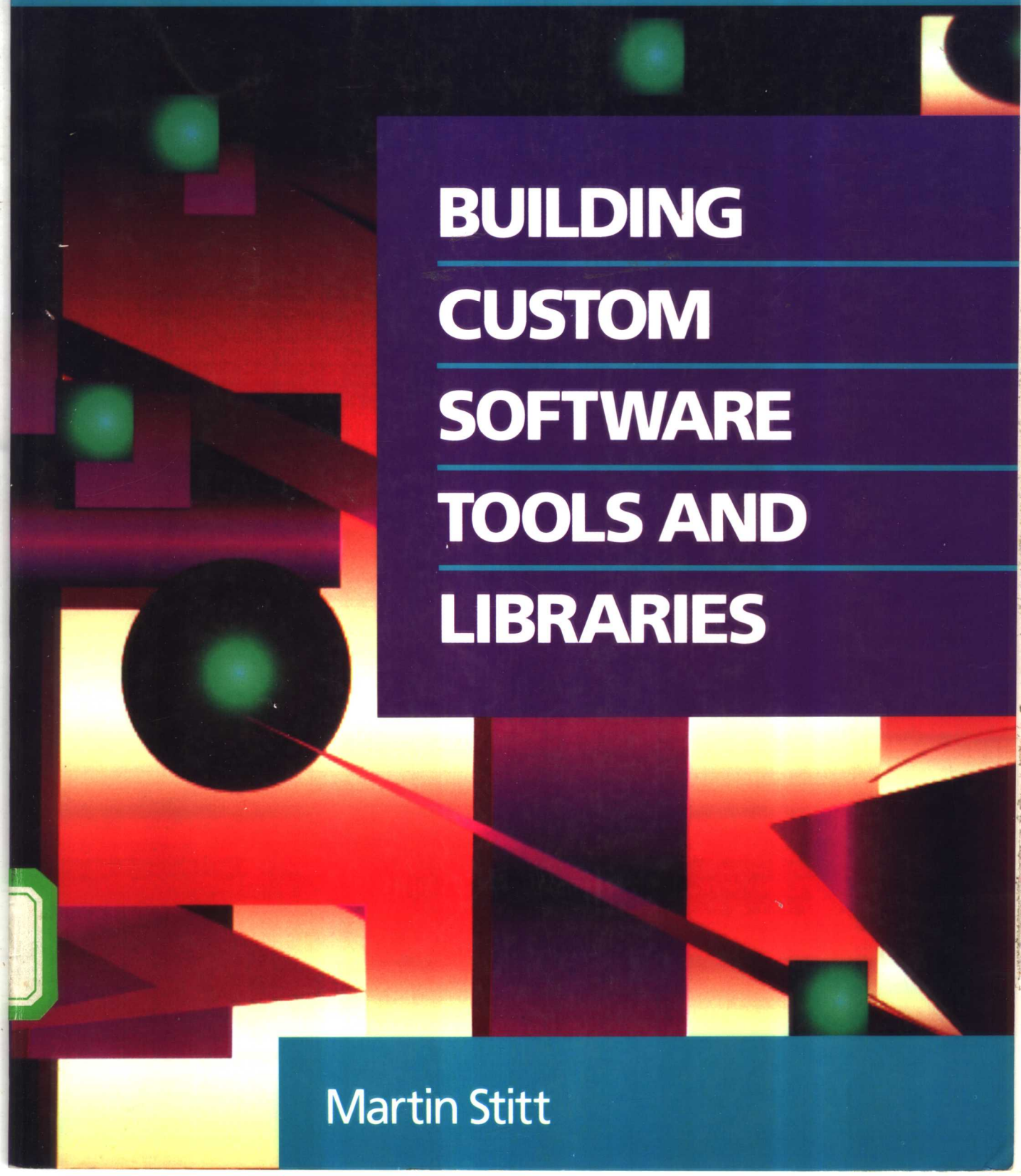
WILEY PROFESSIONAL COMPUTING



# BUILDING CUSTOM SOFTWARE TOOLS AND LIBRARIES

Martin Stitt

# BUILDING CUSTOM SOFTWARE TOOLS AND LIBRARIES

*Martin Stitt*

John Wiley & Sons, Inc.
NEW YORK / CHICHESTER / BRISBANE / TORONTO / SINGAPORE

In recognition of the important of preserving what has been written, it is a policy of John Wiley & Sons, Inc. to have books of enduring value published in the United States printed on acid-free paper, and we exert our best efforts to that end.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc. is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional service. If legal advice or other expert assistance is required, the services of a competent professional person should be sought. FROM A DECLARATION OF PRINCIPLES JOINTLY ADOPTED BY A COMMITTEE OF THE AMERICAN BAR ASSOCIATION AND A COMMITTEE OF PUBLISHERS.



**Library of Congress Cataloging-in-Publication Data**

Stit , Martin
Building custom software tools and libraries / Martin Stitt.
p. cm.
Includes index.
ISBN 0-471-57914-9 (alk. paper : disk). — ISBN 0-471-57915-7 (pbk.). — ISBN 0-471-57916-5 (disk)
1. Computer software–Development. 2. Utilities (Computer programs). I. Title.
OA76.76.D47S47 1993
005.1—dc20 92-33072
CIP

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

# TRADEMARK ACKNOWLEDGMENTS

## ABOUT THE AUTHOR

Martin Stitt began working as an electronics technician in 1972. His first brush with computers came when he designed and hand-built an 8080-based system. Martin progressed from there to work as a systems design engineer, developing microprocessor-based control systems for data communications, energy management, and data acquisition.

Martin now works as a contract software engineer and is involved in such projects as a multitasking/multiuser operating system, X.25, and NETBIOS communications drivers, database applications, CASE systems, debugging tools, and, of course, library development.

His first book *Debugging: Creative Techniques and Tools for Software Repair,* was also published by John Wiley & Sons, Inc. He has written articles on software development published in *PC Tech Journal, Dr. Dobb's Journal, Computer Language, C User's Journal,* and *Tech Specialist.*

# PREFACE

This is a book for those of you who need to write software utilities. You might be an engineer in need of a custom software tool to model the characteristics of a new design. You might be a scientist who requires a custom tool to interface a piece of prototype laboratory equipment. You might be a print-shop technician in need of a custom utility to translate foreign typesetting codes to your system. You may be a programmer in need of a custom tool to create data sets for an application you are testing. The possibilities are endless!

I wrote this book to show how custom software tooling can be created in an efficient manner.

This is not a book on "How to Program in Assembler" or "How to Program in C" (or in any other specific language, for that matter). Books of that sort are plentiful. Nor is it a book that presents a specific library of tool-building components, requiring you to wade through pages and pages of source code listings.

This book addresses the nuts-and-bolts issues involved in the design and implementation of custom tools, including the following:

- Different forms tools can take
- User interfaces
- System interfaces
- Tool-to-tool interfaces
- How compilation and linking work
- The design, construction, and maintenance of function libraries
- Documentation strategies for tools and library modules
- File processing

Library techniques are covered because they make reusable software possible. If you aren't familiar with the design and maintenance of a function

library, then, as you build more and more tools, you either have to reinvent certain wheels or you end up using other, less efficient means to share code.

One of my goals in writing this book was to balance the presentation. When a book on software development is too general, readers must do additional research to attain knowledge that they can apply in their work. Conversely, a book with a narrow focus, such as just covering a certain language or specific brand of compiler, becomes dated quickly.

This book generalizes by presenting its program code in pseudocode form first. The action chart diagramming method is used to impart a graphic clarity to this pseudocode. This approach provides significant detail about the function of the software in an easy-to-read and language-independent manner. (See Appendix A for an explanation of this pseudocode method.)

Many of the pseudocode presentations are followed by an actual implementation in C or assembler. For your convenience, all code examined in this book is available in both source and binary form on the Companion Diskette.

A combination of C and assembler is used in the templates and library functions on the Companion Diskette. The C portions were developed in standard C using Borland C++ version 2.0; the assembler portions were developed using Microsoft's MASM 5.1 and are compatible with Borland's TASM.

The program templates are designed to produce drivers and utilities composed of various combinations of C and assembler code. Device drivers and TSRs can be written entirely in assembler or in a mixture of assembler and C. Stand-alone utility programs are supported as only assembler, only C, or a combination. The program library functions are available to all configurations.

Because of the generalized presentation of the code in the book, implementing this book's techniques in languages other than C should be an easy task. A working knowledge of the MS-DOS operating system and its system-level interface is presumed.

## THE COMPANION LIBRARY

The Companion Library contains the following types of functions:

Parsing Logic
  Parse fixed and switch parameters
  Support response files
Console Input Processing
  Get keys
  Perform background processing while waiting for key input

Check key input for hot keys
Simulate key input
File Processing Logic
Read and process each line of a file
Find first and find next file in a directory
Traverse directory tree on a disk
Stack-Switching Logic
Switch to local stack
Switch back to caller's stack
Pass call on to previous vector holder
String/Memory Logic
Copy blocks of memory
Fill memory
Copy/fill strings
Measure the length of strings
Check for character inclusion in a string
Display Logic
Display characters and strings
I/O Logic
Input a byte from a port
Output a byte to a port
Conversion Logic
Convert binary to decimal ASCII
Convert binary to hexadecimal ASCII
Convert text ASCII to hexadecimal
Convert character to upper case
Miscellaneous Logic
Derive a program's home path
Enable and disable interrupts

# CONTENTS OVERVIEW

# CONTENTS

# CHAPTER 1
# Basic Tool Design

In this chapter we'll take a look at some of the basic aspects of tool design. Because building tools and libraries are often interdependent practices, you'll find that this chapter on tool design also discusses libraries and the chapters on library design cover the construction and design of tools as well.

This may seem like a lot of bouncing back and forth. At times, it may even seem like we're building a house both from the ground up and from the top down. In fact, that is precisely what is being done, such is the nature of interdependence.

## TYPES OF TOOLS

### The Standalone Utility

The standalone utility is the most familiar and common form of program. Most likely, you have used this type of program many times. Its code and data are contained within a file of the `.COM` or `.EXE` type. If you've assembled or compiled the obligatory "Hello world" sample program in your favorite language, then you've already made a standalone utility.

Standalone programs can be invoked manually from the command line or from within a batch file, or they can be executed as a child process of another utility program, such as a command shell, that allows you to start programs by pointing to their name in a list or by clicking on an icon. This type of tool can be as simple as the TREE utility included with MS-DOS, or it can be a more interactive tool—one that presents menus and prompts such as a spreadsheet or a CAD package. This book focuses on the development of