

July 1993
\$4.50 US
Canada \$5.95

Windows™/DOS



DEVELOPER'S JOURNAL

Vol. 4, No. 7

Software Tools

- A C String Extractor
- A Configurable Menu-Based User Interface

PLUS:

- Visual Implementation: Shadowed Popup Windows
- Buffered Window Redraws
- 64Kb Edit Controls; Turning Modal Dialogs into Modeless Dialogs
- Setting and Removing Volume Labels
- Product Spotlight: Windows Installation Utilities
- Adding a Path Finder to BASIC.



Windows™/DOS

DEVELOPER'S JOURNAL



Cover photo by Frithjof Spalder
© The Image Bank/Frithjof Spalder 1990

Software Tools

- Extracting Strings from C Programs** **6**
Hard-coded strings are a major impediment to creating a foreign-language version of your software. This program helps you move those hard-coded string constants out of your C program.
 William F. Dudley, Jr.
- Simplified Windows User Interfaces** **19**
Many small Windows programs do not require a full-blown custom window and message handler. Here's an example of a Windows program whose user interface is provided by menus rather than a window.
 Al Williams

Features

- BASIC Path Finding** **27**
BASIC can't find the path to the directory of the running executable — here's an assembled function that does the job.
 Murray L. Lesser
- Writing a Control Panel Applet for Windows NT** **33**
Learn how to create an NT Applet, and acquire some tips on using NT DLLs and the NT Registry along the way.
 Paula Tomlinson
- Visual Implementation: Shadowed Popup Windows** **49**
Some Windows software (like WinHelp) uses shadowed popup windows, but Windows provides no direct support for this window style. This article provides complete code for shadowed popups.
 Steven Palmer
- Buffered Redrawing** **55**
Windows requires you to repaint on demand, whenever something else covers your window. Here's how you can implement your own "backing store," however.
 Bruce Graves

Product Spotlight

- Windows Installation Utilities** **63**
A look at two tools for creating Windows installation software: Microsoft's Setup Toolkit and InstallSHIELD from Stirling Technologies, Inc.
 Victor R. Volkman

Columns

- Windows Questions and Answers** Paul Bonneau **77**
A summary of the edit control memory management problem and its solutions. A clever and clean technique for turning modal dialogs (such as the Common Dialogs) into modeless dialogs.
- Tech Tips** Leor Zolman **87**
How to set and delete volume labels with DOS XFCB functions. GetInstanceData(): a better solution for building single-instance Windows applications. Using dynamic format strings with printf().

Departments

- From the Editor** **4**
- Source Code Availability** **68**
- Call for Papers** **83**
- New Products** **95**
- Readers' Forum** **99**
- Developer's Marketplace** **100**
- Advertiser Index** **104**

Next Month Interdependent Menu Groups

WINDOWS/DOS DEVELOPER'S JOURNAL (ISSN 1059-2407) is published monthly by R&D Publications, Inc., 1601 W. 23rd St., Suite 200, Lawrence, KS 66046-2743, (913) 841-1631. Second-class postage paid at Lawrence, KS and additional mailing offices. **POSTMASTER:** Send address changes to WINDOWS/DOS DEVELOPER'S JOURNAL, 1601 W. 23rd St., Suite 200, Lawrence, KS 66046-2743. **Subscriptions:** Annual renewable subscriptions to WINDOWS/DOS DEVELOPER'S JOURNAL are \$29 US, \$53 Canada and Mexico, \$64 overseas. Payments must be made in US dollars. Make checks payable to Windows/DOS Developer's Journal. GST (Canada): #129065819

Extracting Strings from C Programs

William F. Dudley, Jr.

Our small CAD software company decided to modify its application programs for users in non-English speaking countries. We were motivated not only by the desire to give non-English speaking users a program with messages in their own language, but also by our linker's refusal to link the application, due to an excessive amount of constant data (strings) in the source code. This article describes the tool we developed to assist in converting our program to an international application.

Removing the Messages

We decided to move almost all of the strings out of the source code and into a separate file. The application would read this file at runtime, stuffing the strings into memory. This not only solved the problem with the linker, it also allowed our foreign distributors to translate the messages file into their native languages, allowing our software to appear much friendlier to our foreign users.

For a large application (40,000 lines) with many embedded messages, the prospect of removing all the messages struck us as tedious at best. Our solution was to write a message extractor, which reads a C source file and, for each message in the file, interactively asks the user if that message should be migrated to the message file. The message extractor remembers all the messages in the message file, so if a message is re-used, the extractor automatically fixes the source code to use only the one instance of the message.

We added only one module to the application - one that finds the message file and reads it, allocating memory for each message as it goes. The application calls this routine at startup as part of the initialization process.

The Message File

The format we chose for the message file is a robust file format that a user can edit with a general-purpose text editor. A "record" holds each message, consisting of three parts:

- A line with the message number and an identifying name (for debugging later on). The identifying name consists of the module name appended with the message number (e.g., *fileio_266*).
- The message itself, possibly on more than one line.
- An *EOT* character (*ctrl-D*, '\004').

William Dudley is a consultant to AT&T Bell Labs Advanced Decision Support Systems, currently working on an Airline Schedule Planning System. He has a Masters in electrical engineering from Cornell University. When not programming for profit or fun, he can be found riding his Norton Commando (on nice days) or working on that or one of his other bikes (on less-nice days). He can be reached electronically at 71631,737 on CompuServe or dud@homxb.att.com on usenet.

The following is a fragment of a message file. The '\004' represents real ^Ds, which don't print.

```
fileio_132
Warning, ROUTER couldn't open file %s for input
'\004'
fileio_133
Error, ROUTER couldn't open tmp file %s
ROUTER exiting to DOS
'\004'
```

The Message Extractor

The message extractor (*makemsg.c* in Listing 1) takes zero or more arguments on the command line. A single argument is interpreted as the name of the message text file; otherwise **MAKEMSG** prompts for it. If you supply two or more arguments, **MAKEMSG** takes the second through last arguments to be the names of C source files in which to look for messages. If you use wildcards for the C source files, they are expanded to the list of files matching the wildcard specification. If less than two arguments are used, **MAKEMSG** prompts for the name of a C source file.

MAKEMSG next tries to open the message text file. If successful, it reads the file into *alloc'd* space (just as the application will do at runtime). Each text record is stored in a *struct* consisting of the string and its index. This index will facilitate a

Listing 1 makemsg.c — Source code for makemsg

```
/* for "Preparing a C application for non-English speaking users"
 * William F. Dudley Jr.
 *
 * Usage:      makemsg msg_file c_file c_file2 . . .
 * Example:    makemsg msg_file ../*.c
 *
 * Note: all the hooks are in place for sorting the strings for
 * faster searching. We are doing linear search for now, but if
 * speed becomes a problem, we can wire it up.
 */
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <errno.h>
#if __TURBOC__
#include <mem.h>
#include <alloc.h>
#elif MSC
#include <memory.h>
#include <malloc.h>
#define MAXPATH 80
#define MAXDRIVE 3
#define MAXDIR 66
#define MAXFILE 9
#define MAXEXT 5
#define EXTENSION 0x02
#define FILENAME 0x04
#define DIRECTORY 0x08
#define DRIVE 0x10
int fnsplit (char *, char *, char *, char *, char *);
void fnmerge (char *, char *, char *, char *, char *);
#endif

#define FALSE 0
#define TRUE 1
#define MAIN 1
#include "makemsg.h"

int loadmsgs (unsigned *, struct msg *, FILE *, int);
FILE *xsrc, *xold, *xnew, *xmsg, *xhdr;
char qsrc[MAXPATH], qold[MAXPATH], qnew[MAXPATH], qmsg[100];
char p[100], q[100], pout[100], psav[100];

int jmsg, jcomment, jquote, jprintf, jif;
int i, j, k, l, m, n;

main (argc, argv)
int argc;
char **argv;
{
    int error = 0;
    unsigned int cnt;
    char qdisk[MAXDRIVE], qpath[MAXDIR], qfname[MAXFILE], qext[MAXEXT];
    char *tok;
    int exists;
    int pathbits, files, argcnt;

    /* Open message file and find highest number. */
    if (argc > 1)
        strcpy (qmsg, argv[1]);
    else {
        printf ("Msg file -> ");
        scanf ("%s", qmsg);
    }
    xmsg = fopen (qmsg, "r");
    if (xmsg == NULL) {
        printf ("No message file %s found.\n", qmsg);
        exit (9);
    }

    /* read existing message file into memory */
    cnt = sizeof (txt)/sizeof (struct msg);
    error = loadmsgs (&cnt, txt, xmsg, 1);
    if (error) exit (error);
    jmsg = cnt;
    fclose (xmsg);
    xmsg = fopen (qmsg, "a");
```

Windows Tools


For C/C++/Pascal & Visual Basic

- Table
 - Column & row split windows
 - Multiple row & column selections
 - Check boxes/ radio buttons/ bitmaps/ editable & combobox column
 - Input Validation
 - Color customization
- Split windows
- Status Bar
 - Auto scrolled text
 - Stretchable field width
 - Colored progress bar
 - Show date/time & key states

Development Tools Kan

Visual Basic version **\$99**

Windows SDK version **\$199**



- 3D Chart
 - Over 30 of 2D & 3D Chart styles
 - Rotation & Scrolling
 - Supports printing & clipboard
- Toolbox
 - Creates buttons from bitmaps or text
 - Supports scrolling
 - 3D buttons w/ color customization
 - Single/ multiple & no-state button groups
- Ribbon / Icon Bar
 - 3D items w/ color customization
 - Supports combobox, text & buttons
- Field Validation
 - Validates date/time/number fields and "PIC" statements
- Meter Control
 - Creates vertical, horizontal and circular gauges with choice of needle or color bar as indicators
 - Linear & Logarithmic scales

With source. Royalties Free. 30-Day MBG.

Consulting & Contract Programming Available

Kansmen Corporation Tel: (408) 988-0634 Fax: (408) 988-0639

Request 109 on Reader Service Card

Listing 1 *continued*

```

if (argc > 2) {
    strcpy (qsrc, argv[2]);
    files = argc - 2;
}
else {
    printf ("Source file -> ");
    scanf ("%s", qsrc);
    files = 1;
}
for (argcnt = 0; argcnt < files; argcnt++) {
    /* first src file is already in qsrc, strcpy not needed */
    /* subsequent src files need to be gotten, however */
    if (argcnt) strcpy (qsrc, argv[argcnt+2]);
    xsrc = fopen (qsrc, "r");
    if (xsrc == NULL) {
        printf ("No source file %s found.\n", qsrc);
        exit (9);
    }

    /* strip off disk and path */
    pathbits = fnsplit (qsrc, qdisk, qpath, qfname, qext);
    fnmerge (qold, NULL, NULL, qfname, qext);
    /* make qnew have no drive or directory, new files made in CWD */
    fnmerge (qnew, NULL, NULL, qfname, qext);
    /* if source is not in another directory,
     * append 'n' & 'o' to file names */
    if (!(pathbits & (DIRECTORY | DRIVE))) {
        strcat (qold, "o");
        strcat (qnew, "n");
    }

    xnew = fopen (qnew, "w");

    jcomment = FALSE; jprintf = FALSE; jif = FALSE;
    while (TRUE) {
        fgets (q, 128, xsrc);
        if (feof (xsrc)) break;

        jquote = 0;
        kquote = 0;
        if (q[0] != '#') {
            for (j = 0; q[j]; j++) {
                if (!strncmp (&q[j], "printf", 6)) jprintf = TRUE;
                if (!strncmp (&q[j], "#if", 3)) jif = TRUE;
                if (!strncmp (&q[j], "#endif", 6)) jif = FALSE;

                if (q[j] == '/' && q[j+1] == '*') jcomment = TRUE;
                if (q[j] == '*' && q[j+1] == '/') jcomment = FALSE;

                if (jprintf == TRUE && jcomment == FALSE &&
                    jif == FALSE && q[j] == '\\')
                {
                    if (jquote == 0) jquote = j;
                    else if (kquote == 0) kquote = j;
                }
            }
        }
        if (jquote && kquote) {
            /* extract the string, save it in psav[], */
            /* expand it into pout[] */
            k = 0;
            for (j = jquote+1; j < kquote; j++) { psav[k] = q[j]; k++; }
            psav[k] = 0;
            for (j = 0, tok = psav; *tok; tok++) {
                if (*tok != '\\')
                    pout[j++] = *tok;
                else {
                    switch (tok[1]) {
                        case 'n': /* we found "\n" */
                            pout[j++] = '\n';
                            break;
                        case 't':
                            pout[j++] = '\t';
                            break;
                        case 'v':
                            pout[j++] = '\v';
                            break;
                        case 'b':
                    }
                }
            }
        }
    }
}

```

Windows & DOS Programming Tools

Sourcer™

New
v5.0

**"Sourcer is the best disassembler
we've ever seen."** PC Magazine

Creates commented source code and listings from binary files. Shows how programs work with detailed comments on interrupts, subfunctions, I/O functions, and more. Supports all instructions to 80486 and V20/V30.

Sourcer provides the best analysis separating code and data. It automatically determines data types, uses descriptive labels for BIOS and PSP data, and links data items across multiple segments.

New version 5.0 makes most DOS EXE and COM files and drivers reassemble perfectly, byte-for-byte identical to the original!

Top professionals depend on Sourcer for the most reliable results with the least effort.

New v2.0 for Windows

**"Sourcer combined with Windows
Source should be mandatory for
looking into Windows Programs."**
Sal Ricciardi PC Magazine

Windows Source™ with Sourcer generates detailed listings of Windows EXEs, DLLs, SYSs, VxDs, device drivers & OS/2 NE files. Labels, by name, export & import function calls, API calls like "GetFreeSpace" and more.

See the many undocumented Windows functions used by professionals to perform tricks that are otherwise impossible.

Comes complete with extra utilities for resource extraction and import analysis. Uses CodeView symbols for improved clarity.

BIOS Source

for PS/2, AT, XT, PC and Clones

The BIOS Pre-Processor™ with Sourcer creates commented listings for any BIOS ROM in your PC. Understand how your specific BIOS works! Adds over 75K of comments specific to your BIOS. Identifies multiple interrupt branches with special labeling like "int_10_video." Fully automatic.

Sourcer-Commenting Disassembler	\$129.95
Sourcer w/BIOS -(save \$10)	169.95
ASMtool 486-Automatic flowcharter	199.95
ASM Checker-Finds source code bugs	99.95
Windows Source-requires Sourcer	129.95
Windows Source & Sourcer-(save \$30)	229.90

Shipping: USA \$6; Canada/Mexico \$10; Other \$18. CA residents add sales tax. © 1993 VISA/MasterCard/COD

30-DAY MONEY-BACK GUARANTEE

1-800-648-8266 order desk



V Communications, Inc.

4320 Stevens Creek Blvd., Suite 275-WD
San Jose, CA 95129 FAX 408-296-4441
408-296-4224

Listing 1 *continued*

```

        pout[j++] = '\b';
        break;
    case 'r' :
        pout[j++] = '\r';
        break;
    case 'f' :
        pout[j++] = '\f';
        break;
    case '0' : case '1' : case '2' : case '3' :
    case '4' : case '5' : case '6' : case '7' :
        i=tok[3]-'0'+(tok[2]-'0')*8+(tok[1]-'0')*8;
        tok += 2;
        pout[j++] = (char)i;
        break;
    case '\\':
        pout[j++] = '\\';
        break;
    default :
        tok--; /* don't skip 2 chars */
        break;
    }
    tok++; /* because we prob. found "\n" */
}
}
pout[j] = '\0';
if (kquote-jquote < 3) {
    fprintf(xnew, "%s", q);
}
else {
    exists = -1;
    /* is string found in existing message file ? */
    for(j = 0; j <= jmsg; j++) {
        if(!strcmp(pout, txt[j].s)) break;
    }
    /* does string exists in message file already ? */
    if (j != jmsg+1) exists = txt[j].index;
    if (exists >= 0)
        printf ("%s: string found in msg file, #d: Fixing.\n", qsrc, exists);
    else {
        /* does user want to move string out of source file ? */

        printf ("\n%s\n", q);
        printf ("\t\tLeave it or Fix it? "); scanf ("%s", p);
        if(p[0] == 'l' || p[0] == 'L') {
            fprintf(xnew, "%s", q);
            continue;
        }
        else if (p[0] != 'f' && p[0] != 'F')
            goto Query;
    }

    if (exists < 0) {
        /* then we haven't seen this string before */
        jmsg++; /* Bump message counter. */
        if (jmsg > MAX_MSGS) {
            fprintf(stderr, "too many strings\n");
            exit (ENOMEM);
        }
        /* now store message in bufr array */
        j = strlen (pout);
        txt[jmsg].s = calloc (j+1, 1);
        if (txt[jmsg].s == NULL) {
            fprintf(stderr, "can't alloc more memory, line %u\n", msg);
            exit (ENOMEM);
        }
        strcpy (txt[jmsg].s, pout);
        txt[jmsg].index = jmsg;
        exists = jmsg;
        fprintf (xmsg, "%d %s %03d\n", jmsg, qname, jmsg);
        fprintf (xmsg, "%s\n\004\n", pout);
    }

    /* Print the line without quoted string. */
    strncpy(p, q, jquote); k = jquote;
    k += sprintf (p+k, "txt[%d]", exists);
    for (j = kquote+1; q[j]; j++) { p[k] = q[j]; k++; }
    p[k] = '\0';
    fprintf (xnew, "%s", p);
    /* And put a comment into the code with the string. */
}

```

Query:

planned enhancement, sorting the strings so that a binary search can be used instead of the current linear one. So far, this revision has proven unnecessary, since a linear search of a few hundred strings doesn't take very long.

Next *MAKEMSG* runs through a loop for each of the C source files specified on the command line. If the source file name has a directory component to its name (i.e., resides in a directory other than the current one), *MAKEMSG* reads the source file, but writes a new source file of the same name in the current directory. If the source file is in the current directory, *MAKEMSG* creates a new source file with name *XXXXn* (e.g., *eclipse.c* becomes *eclipse.cn*). If the text extraction goes without problems, then the old source file is renamed to *XXXXo* (e.g., *eclipse.co*) and the *XXXXn* file gets the original name of the old source file.

The next part of the loop decides what constitutes a string and asks the user to "Fix it or Leave it?" These rules you will most likely want to customize for your application. Our rules are fairly simple: the string can't be bracketed by *#if/#endif* (so as to ignore most debug *printf*s), cannot be in a comment, and must start on a line with a "printf" (which encompasses *sprintf*, *fprintf*, and *vsprintf*).

Once *MAKEMSG* detects a candidate string, it searches the existing message database for a copy of it. If one is found, the source file is automatically fixed to use the existing stored string. If the string is not in the database, the user is asked to "Fix it or Leave it?" If "Fix," *MAKEMSG* adds the string to the message base, appends the new text record to the end of the message file, and fixes the source file to use the proper entry in the (future) *txt[]* array. If "Leave," *MAKEMSG* simply copies the source file to the new file and cycles back to the top of the *while* loop.

The Message Loader Module

The message loader module, shown in Listing 2, searches down the DOS *PATH* for the message file. After successfully opening the file, the module reads each record, checks that its number agrees with the loop counter (the file corruption test), allocates the memory for the string, copies the string to that

memory location, and sets the pointer in the `txt[]` array to point to the string.

The loader returns to the caller the number of the last message read, so that if the integrity check fails, a message is printed on the console telling the user at what line in the `txt` file the error occurred.

Creating Messages

Once you have decided to make your messages translatable to other languages, you have new restrictions on how you may generate your messages. For example, the `printf` construct

```
intf("There %s %d via%s in this job.\n",
    (via_count1) ? "are" : "is", via_count,
    (via_count1) ? "s" : " ");
```

which reports the number of items in a job, will not translate well into other languages. You will be lucky if the target language makes plurals as English does. You should instead have two messages – one for the singular, and one for the plural.

In general, when writing messages, don't do anything tricky using English grammar, spelling, or parts of speech.

Another consideration is the size of the message buffers. Our experience suggests that if you plan on producing a German version of your program, make sure that the buffers into which you copy your messages (using `sprintf()` or `strcat()`, for example) are at least twice as long as needed by the

Software Developers ...

Ask Corel, AT&T,
EXXON, Sharp,
Inset Systems, and
a thousand
others ...

why they picked
LEADTOOLS for
their image
application.



Original 39 MB compressed
to 163 K, 239:1.

They'll say from
document to true color
imaging, **LEAD Technologies** is innovative.

LEAD pioneered image compression technology that achieves compression ratios of over 200 to 1, constructed tools for quick integration of images into any application, and built a toolkit with a reputation for speed! Call for a **free evaluation diskette** to see for yourself.

LEAD Technologies, Inc.
1-800-637-4699 • Fax 704-548-8161

Request 161 on Reader Service Card

Listing 1 continued

```

        m = 60 - strlen(psav);
        while (m > 7) { fprintf(xnew, "\t"); m -= 8; }
        while (m > 0) { fprintf(xnew, " "); m -= 1; }
        fprintf(xnew, "/* >> %s << */\n", psav);
    }
}
else fprintf(xnew, "%s", q);
}
fclose(xsrc);
fclose(xnew);
if (!(pathbits & (DIRECTORY | DRIVE))) {
    unlink(qold);
    rename(qsrc, qold);
    rename(qnew, qsrc);
}
}

fclose(xmsg);
jmsg++;
xhdr = fopen("menuload.h", "w");
fprintf(xhdr, "char *txt[%d];\n", jmsg);
fclose(xhdr);

exit(0);
}

#ifdef MSC
/* this function is in the Turbo-C library */

int fnsplit(path, disk, dir, name, ext)
char *path, *disk, *dir, *name, *ext;
{
char bdisk[MAXDRIVE], bdir[MAXDIR], bname[MAXFILE], bext[MAXEXT];
char *cp1, *cp2;
register int i;
int result;

i = 0;
cp1 = strchr(path, ':');
if (cp1 == NULL) cp1 = &path[-1];
else for( ; &path[i] <= cp1 ; i++) bdisk[i] = path[i];
bdisk[i] = '\0';
cp2 = strchr(path, '\\');
i = 0;
if (cp2 == NULL) cp2 = strchr(path, '/');
if (cp2 == NULL) cp2 = cp1;
else for( cp1++; cp1 <= cp2 ; i++, cp1++) bdir[i] = *cp1;
bdir[i] = '\0';
for( i = 0, cp2++; ((*cp2)&&(i < 8)&&(*cp2!='.')) ; i++, cp2++) {
    bname[i] = *cp2;
}
bname[i] = '\0';
for( i = 0 ; ((*cp2)&&(i < 4)) ; i++, cp2++) {
    bext[i] = *cp2;
}
bext[i] = '\0';
if (disk!=NULL) strcpy(disk, bdisk);
if (dir!=NULL) strcpy(dir, bdir);
if (name!=NULL) strcpy(name, bname);
if (ext!=NULL) strcpy(ext, bext);
result = (strlen(bdisk)) ? DRIVE : 0 ;
result += (strlen(bdir)) ? DIRECTORY : 0 ;
result += (strlen(bname)) ? FILENAME : 0 ;
result += (strlen(bext)) ? EXTENSION : 0 ;
return(result);
}

void fnmerge(path, disk, dir, name, ext)
char *path, *disk, *dir, *name, *ext;
{
    sprintf(path, "%s%s%s%s", path, disk, dir, name, ext);
}
#endif

/* End of File */

```


Listing 2 txtload.c — Module to load message text into memory

```

/* compiles with MSC 4.0, TCC 2.0, or Watcom 386 8.0
 * makefile sets MSC true when using MSC 4.0 compiler.
 */

#include <dos.h>
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#if __TURBOC__
#include <mem.h>
#include <alloc.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#else MSC
#include <memory.h>
#include <malloc.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#endif

#define MAIN 0
#include "makemsg.h"

char nothing[] = "";

#define MENBUFLEN 256

/* load messages into calloc'd array.
 * typical usage:
 unsigned cnt;
 int error, line;
 struct msg *txt[139];
 FILE *menufile;
 menufile = fopen("ROUTEMSG.TXT","r");
 cnt = sizeof (txt)/sizeof (char *);
 error = loadmsgs (cnt, txt, menufile, load, &line);
 if (error) {
 printf ("Error at line %d in ROUTEMSG.TXT.\n", line);
 exit (EFORMAT);
 }
 * return number of messages in *cnt.
 * return error condition or 0 if OK.
 */

/* ***** LOADMSG ***** */
int loadmsgs (cnt, arp, mfile, lod)
 unsigned int *cnt;
 struct msg arp[];
 FILE *mfile;
 int lod;
 {
 int j, k, row;
 char lbuf[MENBUFLEN];
 char bufr[MENBUFLEN];
 int error=0;
 int col=-1;
 for (k = 0 ; k < *cnt ; k++) {
 if (NULL == fgets (lbuf, 128, mfile)) { *cnt = k-1; break; }

```

STOMP YOUR WORST C BUGS. EFFORTLESSLY.



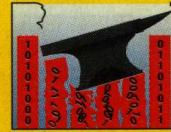
HEAP CORRUPTION



MEMORY
LEAKS



MEMORY OVERWRITES



DATA CORRUPTION

Toughen up against bugs

If you're seriously interested in writing the best, most reliable C programs you can, take a giant step to increase your productivity with MemCheck. Discover how easy it is to stomp the worst, most pervasive bugs in the C language. With no source code changes, and no special compiler options, MemCheck is a rock solid defense against a whole cadre of common memory errors.

Silent and deadly

MemCheck operates completely automatically, interacting with the developer only to pinpoint memory errors by exact source file and line number. Otherwise it's as quiet as an eagle on the wind. Without your

even having to know it's there, MemCheck is a potent weapon in all phases of development.

Seamless integration

MemCheck integrates seamlessly into your Windows or DOS C projects in 15 minutes. You'll pinpoint overwrites on GlobalAllocs, LocalAllocs, and all linear memory allocations supported by your compiler (malloc, calloc, etc). Find unfreed memory, heap corruption, and more. Up to *30 times faster* than other debuggers trying to do the same things.

And you can use MemCheck with C++ to keep your C++ apps in tip-top shape.

Easy to use

Developing reliable software is challenging for the best of us. Using MemCheck brings big payoffs for novice and expert alike in quality and time savings. But you be the judge—your satisfaction is guaranteed. Call 1-800-933-3284 (1-800-WE-DEBUG) and take the *fastest, easiest approach* to stomping the whoppers of C bugs.

MemCheck comes ready to run — be sure to specify compiler and platform!

DOS \$139
WINDOWS \$179 \$99
MACINTOSH \$179 \$99

SPECIAL
THIS
MONTH!

SPECIAL BUNDLES!
DOS MASTERS PACK \$199
for Microsoft C AND Borland C... save \$80
WINDOWS GURU PACK \$159
for Microsoft C AND Borland C
PC POWER PACK \$199
any DOS version + any Windows version

THE EXPERTS' CORNER

"MemCheck is worth its weight in gold."

-- David Thielen, author of 'NO BUGS: Delivering Error-Free Code in C and C++'

MEMCHECK™

ADD EFFORTLESS RELIABILITY TO YOUR C APPLICATIONS

1-800-WE-DEBUG



WE USE AND SHIP QUALITY RECYCLED MATERIALS.

StratosWare Corporation 1756 Plymouth Road, Suite 1500 • Ann Arbor, MI 48105 • 1-800-WE-DEBUG • International (313) 996-2944 • Fax Lines (313) 996-2955 & (313) 747-8519

☐ Request 153 on Reader Service Card ☐

Listing 2 *continued*

```
row = atoi(lbuf);
if (k!=row) { error++; break; }
if (k!=(col+1)) { error++; break; }
col = row; /* save row for monotonicity check */
memset (bufr,0,MENBUFLLEN);
if (NULL == fgets (lbuf, 128, mfile)) { error++; break; }
while(lbuf[0] != 4) {
    if(bufr[0]) strcat (bufr, "\n");
    if (lbuf[0] == '\n') strcat (bufr, "\n");
    else {
        lbuf[strlen (lbuf)-1] = '\000';
        strcat (bufr, lbuf);
    }
    if (NULL == fgets (lbuf, 128, mfile)) { error++; break; }
}
/* now store message in bufr away */
if (lod) {
    j = strlen (bufr);
    arp[k].s = calloc (j+1, 1);
    if(arp[k].s == NULL) {
        fprintf(stderr,"can't alloc more memory, line %u\n",k);
        return(ENOMEM);
    }
    strcpy (arp[k].s, bufr);
}
else arp[k].s = &nothing[0];
arp[k].index = k;
}
return (error);
}
/* End of File */
```

English versions. (The German word for "bus," as in "electrical bus," is "verbindungsbündel.")

Summary

If you sell your programs to non-English speaking users, you should address the problem of communicating in their language, if you haven't already done so. The module and program presented here should help you to modify your application so that it can load its messages from an external file, which can then be translated into other languages.

The code is designed to compile under three compilers: MSC 4.0, TCC 2.0, and Watcom 386 8.0. This work does not consider the problems that arise with languages which need 16-bit characters, such as Chinese or Japanese. □

Listing 3 *makemsg.h — makemsg header file*

```
/* for "Preparing a C application for non-English
 * speaking users"
 * William F. Dudley Jr.
 */

#define MAX_MSGS 10000
#if !MAIN
extern
#endif
    struct msg {
        char *s;
        unsigned index;
    } txt[

#if MAIN
        MAX_MSGS
#endif

];

/* End of File */
```

Defect management made easy with Defect Control System for Windows



The award-winning defect manager that organizes and monitors bug reports.

Robust query and report facilities help you deliver quality software on time.

Notification features and change histories keep your team informed.

Easy customization means DCS won't change the way you work.

Call now for your free demo disk and product information: 719-598-3713

SOFTWARE EDGE
The Leader in Defect Management

4420 Laven Way
Colorado Springs, CO 80920



□ Request 174 on Reader Service Card □

Listing 4 *makemsg makefile*

```
# $* is the target w/o suffix
MAKE_TMP = $(TMP)
OBSJS = makemsg.obj txtload.obj setargv.obj

CC = tcc
# -O optimize TCC for size
# set D=-v for TDebug
#D=-O
D=-v
TCC_OPTS = -mc $(D)
CC_OPTS = $(TCC_OPTS)
ASM = tasm

.SUFFIXES: .exe .obj .c .asm

.AFTER:
    @ beep

.c.obj:
    tcc $(CC_OPTS) -c $*.c

.asm.obj:
    $(ASM) /mx $*;

makemsg.exe:    $(OBSJS)
                $(CC) $(CC_OPTS) -e$*.exe $(OBSJS) setargv.obj

makemsg.obj:    makemsg.c makemsg.h

txtload.obj:    txtload.c makemsg.h
```