BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-92-M14

"XComment: An Interactive Documentation Tool"

by

Christopher A. Wood

XComment:
An Interactive Documentation Tool

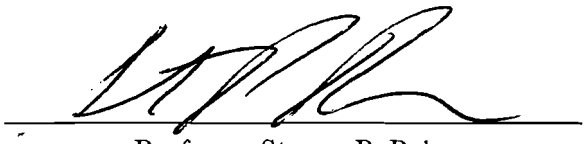
Christopher A. Wood

Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the
Degree of Master of Science in the Department of Computer Science
at Brown University

May 1992

This research project by Christopher Wood is accepted in its present form
by the Department of Computer Science at Brown University
in partial fulfillment of the requirements for the Degree of Master of Science.



Professor Steven P. Reiss
Advisor

5/3/92
Date

Contents

1 Introduction

XComment is an extension of the autocommenter program called **autoc**, which was written at Brown University by David Fedor in 1990. **Autoc**'s behavior was such that it would load a C, C++, or Pascal file, scan it, and insert header comments throughout the file according to several switches that the user could set on the command line. **XComment** provides the user with a Motif Graphical User Interface, which allows the user to interactively insert comments anywhere in the file, in addition to having the program generate header comments on its own.

The scanning of the input file is done using much of the existing code from **autoc** to obtain the starting positions of the functions within the file. All of this code is contained in *xcommentBase.c*, which was formerly named *autocbase.c*. New code is also found in this file, dealing with the interpreting of the file. The user interface code, as well as other new code, is found in *xcommentFile.c* and *xcommentUI.c*. The global include file is called *xcomment.h*, formerly *autoc.h*. The actual explanations of each of these modules appear in Section 3.

2 The User Interface

The user interface for **XComment** was built using Motif. There are three popup windows that are worth talking about here.

2.1 The File Selection Window

The first window that the user will see depends on whether or not he types in a filename on the command line. If he does not, XComment pops up a menu of pushbutton widgets, referred to as the File Selection Window, which looks like the following:

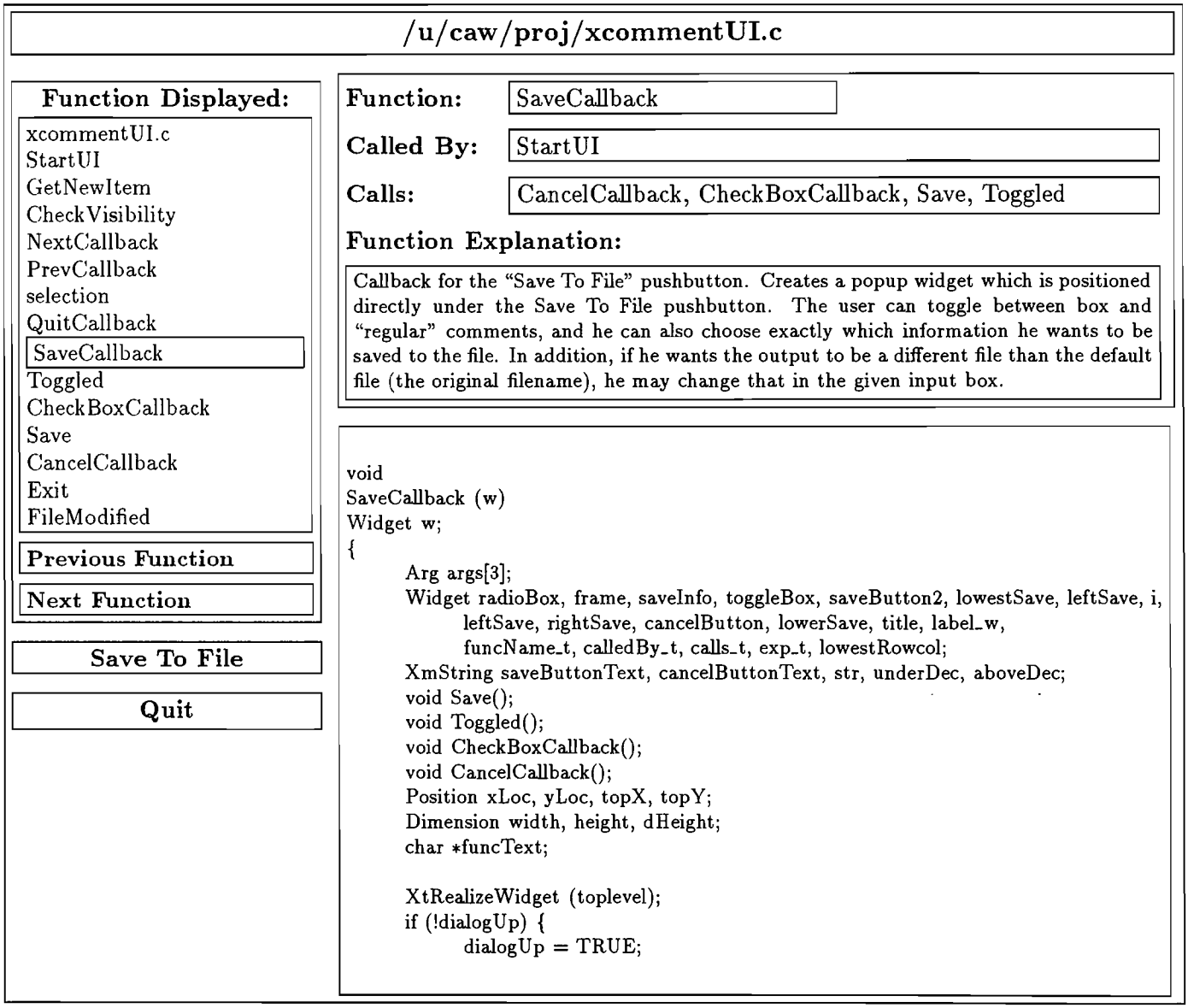
Open C File
Open C++ File
Open Pascal File
Quit XComment

Upon selecting one of the pushbuttons, a standard File Selection Box will pop up, displaying those files in the current directory with the appropriate extension. There is also a filter in case the user would like to view other directories' contents.

On the command line, if the user types in the name of the file that he would like to comment, the File Selection Window will be bypassed and the Main Window will appear. This is the same window that the user will see if he had first gone through the File Selection Window.

2.2 The Main Window

The Main Window is a popup widget if the File Selection Window has been activated; otherwise it is the top level application widget. It provides the user with a list of functions or procedures along the left side; several widgets which hold parts of the header comment of the current function in the top portion of the right side; and also the program text which defaults to the starting position of the current function in the bottom portion of the right side. It looks something like the following picture:



2.2.1 The Function/Procedure List

Along the left side of the window, there is a sequential list of all of the functions in the input file. If there is not a "program" statement at the start of the file (as may be the case in a Pascal program), the first name in this list is the name of the file. This gives the user the opportunity to enter some comments in the Explanation Text widget, which will be placed at the very beginning of the file.

The list widget can visibly display up to twenty function names. If there are more than twenty functions, a scrollbar is attached on the right side to let the user scroll to the "invisible" function names. By clicking on a function name in this list, the user will cause that function to be displayed in the lower right portion of the window, with its comment information appearing in the upper right portion.

Underneath this list, there are two useful pushbuttons: One labeled "*Previous Function*", and one labeled "*Next Function*" (In the case of Pascal programs, the "Function" is replaced by "Procedure", as is the case in many other situations in this window.) These buttons simply automatically advance the function name to the next or previous function in the list. Even if the next or previous function is not visible, these buttons still work, and the list will scroll accordingly if needed.

2.2.2 Function Comment Information

In the upper right portion of the window are widgets which contain important information about the selected function. XComment automatically inserts the function name into the top entry field. During the scanning of the file, it stored every function call that each function made, and that information is contained in the next two entry fields, the "**Called By**" and the "**Calls**" widgets. Note that the "Calls" entry field only contains those functions *within the current file* that are called by the present function. These three fields are not manually modifiable in any way.

If the input file contains header comments which includes some of the above information, XComment does its best to trash these lines so the user doesn't have to deal with redundant information later after it's saved to the file. This information is suppressed only if a string in the comment exactly matches one of the following: "Function:", "Procedure:", "Program:", "Called By:", and "Calls:". So obviously if the user always uses XComment, he will have no problems with repeating information in the header comments. Otherwise, these comments will show up in the Function Explanation Text Widget, which could be deleted if the user desires.

The final text entry field is the **Function Explanation text widget**. This one is modifiable. If there existed any header comments for the selected function before loading the file, they will appear here and not in the program text. (They have been extracted and inserted into this widget.) If the user desires to delete this information, he may do so. He can also add to it, or modify it in any way. If the user leaves this function and

then later returns, the changes that were made will still be there, right up until he quits the program. Any changes made will naturally be saved along with everything else when the user saves the file. There is also a scrollbar along the right side of this widget, which is especially useful, since only five lines are visible at one time.

2.2.3 The Program Text

In the lower right portion of the window is an 80 x 23 text widget which contains the entire text of the input file. There are scroll bars along the bottom and right side of this widget in case the user needs to look at other parts of the file, or if he wants to comment within the program text. When the user clicks on one of the function names in the function list, the top visible character in this widget will automatically scroll to the starting position of the selected function. This may be useful to the user in case he wants to look at the actual function while typing in the function explanation.

Modifying the Program Text: Whenever the user does something to change the program text (e.g. inserting or deleting comments or code), a callback is made which causes the starting positions of every function which follows the one that's being modified, to be adjusted accordingly. Therefore, the user is free to modify any part of the file at any time, without affecting the placement of the comments upon saving the file, nor affecting the ability of XComment to find the starting position of each selected function for placement in the Program Text Widget.

However, There are a few things that the user *should not* do to the program text while it's in XComment...

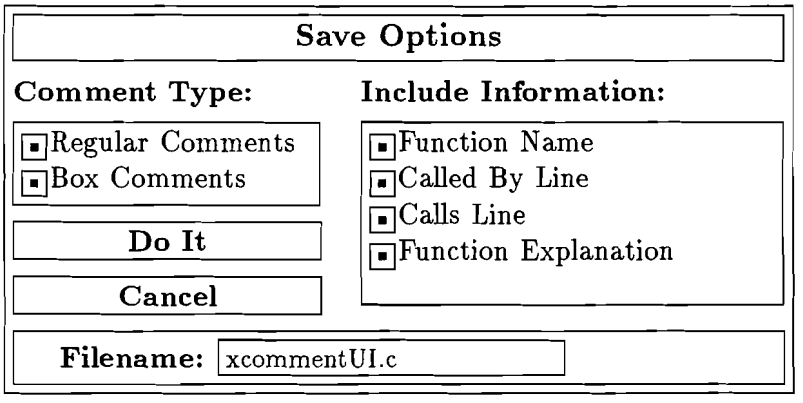
- **Don't delete entire functions.** This is not a total disaster; however make sure you look at your file after you save it. What will probably happen is the deleted function's header comment will be inserted before the previous function (in addition to that function's comments).

- **Don't delete code which may contain calls to other files.** If you do, the information in the "Called By" and "Called" fields may be incorrect. This is not meant to be a full-blown program editor. If you are modifying *code* here, maybe you should load your program into your favorite editor and finish it up before you document it.

- **Don't move functions around.** If for some reason you'd rather have function *foo* at the end of the file instead of at the beginning, don't move it here. The file is only scanned once *before* you even see it in the Main Window. It won't rescan it to update itself. If you really want to do this, you could exit and then reload it.

2.3 The Save Option Window

When the user decides that he's documented enough and is ready to save the results to the file, he obviously presses the "**Save To File**" pushbutton, located under the function list on the left side of the Main Window. XComment then pops up a new dialog box which gives the user a few options for saving the file. This window will appear directly under the Save To File pushbutton, and it looks something like this:



The left half of the Save Options popup dialog box contains a toggle between "regular" and "box" (default) comments. By clicking on the radio button to the left of one of these comment styles, it will become the active style. The right half contains four checkboxes which affect what is written back to the file over each function. They all default to "on". By turning these on and off, the user can come up with sixteen different possibilities of header comment content.

2.3.1 Using XComment as an Uncommenter

If the user loads a commented program from which he would like to delete all header comments, all he would have to do is turn all of the checkboxes off, and he would end up with a program without any header comments.

2.3.2 Saving the File

If the user wants to save the new file to a different name, he may do so using the bottom entry field in the Save Options window. The filename defaults to the original filename, so any saving without modifying the name will result in the original version being written over with the new version.

2.3.3 Format of Comments

The comment format, assuming all checkboxes are on, is the following:

Box Comment Style:

```
/************************************************************************ */
/* Function:     SaveCallback                                             */
/*                                                                        */
/* Called By:    StartUI                                                  */
/* Calls:        CancelCallback, CheckBoxCallback, Save, Toggled          */
/*                                                                        */
/* Callback for the ''Save To File'' pushbutton.  Creates a popup widget  */
/* which is positioned directly under the Save To File pushbutton.  The   */
/* user can toggle between box and ''regular'' comments, and he can also  */
/* choose exactly which information he wants to be saved to the file.     */
/************************************************************************ */
```

"Regular" Comment Style:

```
/*
 * Function:     SaveCallback
 *
 * Called By:    StartUI
 * Calls:        CancelCallback, CheckBoxCallback, Save, Toggled
 *
 * Callback for the ''Save To File'' pushbutton.  Creates a popup widget
 * which is positioned directly under the Save To File pushbutton.  The
 * ...
 */
```

3 Module Explanations

There are three source files for XComment. They are as follows:

xcommentFile.c	File Selection UI, Gets ready for scanning
xcommentBase.c	File scanner, comment parser, writes comments to file
xcommentUI.c	Main Window UI, Save Info Window, callbacks, save routine

All of the major functions are explained to some degree under each module. Most functions are at least mentioned, but some of the unimportant ones are not included in the interest of brevity and clarity. The ones listed are also not necessarily in sequential order; rather they are grouped according to what they do.

3.1 xcommentFile.c

3.1.1 Opening the File

GetDefaults
ReadFile
main
Open
UpdateMessage
open_file

These functions are concerned with doing what is necessary to get a file loaded, and calling DoFile (in *xcommentBase.c* to get things started. Main obviously starts things off, and if the user provided a filename on the command line, it doesn't bother calling Open or open_file; it just calls GetDefaults (which sets beginComment & endComment) and the fileType, before calling DoFile. ReadFile is called to make sure the file is valid. UpdateMessage is called to refresh the popup dialog box which just lets the user know that the file is being scanned (only if he went through the File Selection Box). The Selection Box is brought up in function Open, and open_file is the callback associated with the "ok" button of the box

3.1.2 Destroying Dialogs

Kill
KillSelectionBox
KillandOpen

Kill is the callback for the "Quit XComment" pushbutton on the File Selection Window, and it causes the program to shut down. KillandOpen first kills the FileSelectionBox when the "ok" button is pressed, then calls open_file to open the selected file. KillSelectionBox is activated when the user presses the "Cancel" button on the File Selection Box, causing the Selection Box to be destroyed.

3.2 xcommentBase.c

3.2.1 Saving the Comments to the File

addblankcommentline
addexplanationtext
addgeneratedlines
getgeneratedlines
getfctinfo
writeheader

These functions are concerned with the saving of the comment headers to the file. They are called by the Save and DoCommentBox routines in *xcommentUI.c.* Basically, writeheader is called first and it writes the function name to the file in between beginning and end comment characters. It then calls addblankcommentline before it's done. Getgeneratedlines writes the "Called" and "Called By" information to the file, again calling addblankcommentline at the end. Finally, addexplanationtext is called to add the explanation text to the file. It has to go a few calculations, such as inserting return characters throughout the file to comply with maximum line length.

3.2.2 Parsing the Incoming Comments

StripOutRedundantInfo
StripOutCommentChars
GetComment
ExtractComments
AdjustStartingPositions

These functions deal with the extraction and parsing of the header comments which were found in the input file. ExtractComments searches backwards from the function declaration line, and if it is successful in finding a comment, it sends it over to GetComment, which kills the comment text from the actual program text, and also adjusts the starting positions of all functions which follow the function that it's working on. After this, the comment is sent to StripOutCommentChars, which does just that. All begin and end comment characters in the comment are destroyed. Finally, the comment is sent to StripOutRedundantInfo, which searches the comment for strings such as "Function:", "Procedure:", "Program:", "Called By:", and "Calls". If any of these are found, the line is killed since we don't need it in the explanation text. Now the comment is ready for insertion into the appropriate explanation text widget, and this is done.

3.2.3 Scanning the File

getfctinfo
AdjustStartingPositions
docontent
dofile
DoFile

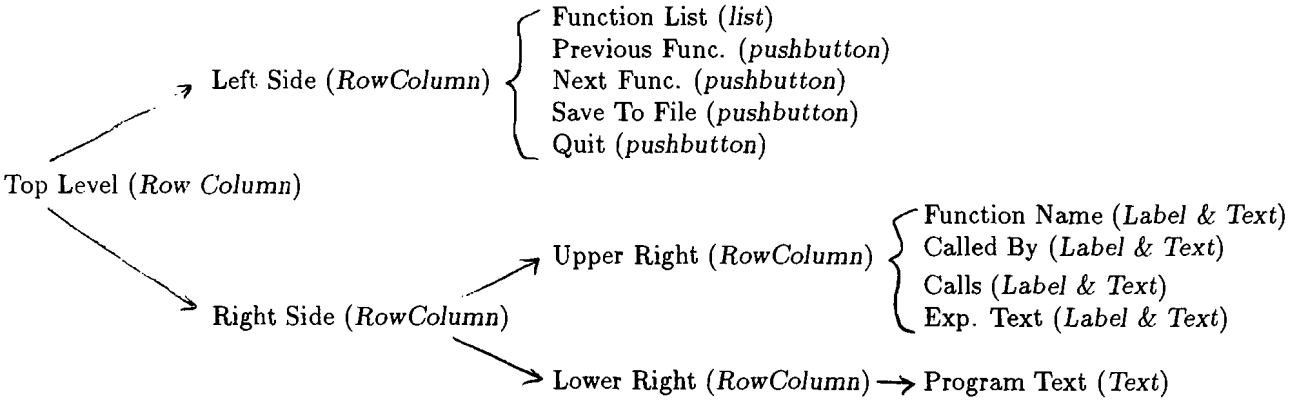
These functions deal with calling XREF to take care of scanning the file. The scanning produces the starting lines and positions, as closely as possible, to the functions or procedures in the file. Sometimes, however, the starting positions are just a little bit off. For example if a C function declaration takes two lines, the old *autoc* would not give an accurate placing of the comment for that function. This is what AdjustStartingPositions is responsible for. There are other cases, such as the starting point of Pascal procedures or C++ functions being one line too early. Most likely, all situations have not been taken care of, but the most frequent ones have.

3.3 xcommentUI.c

3.3.1 The Main Window UI

StartUI

This function contains all of the widgets which make up the XComment Main Window. Following is a representation of the hierarchy of these widgets. In parentheses for each widget is its widget type.



3.3.2 Main Window Callbacks

GetNewItem
CheckVisibility
NextCallback
PrevCallback
selection

These are all concerned with the function list widget. NextCallback and PrevCallback are obviously the callback functions for the Next and Previous pushbuttons. They both call CheckVisibility to make sure the selected item is visible. When these are called, so is selection, which is also called when the user clicks on one of the functions. Selection also calls GetNewItem, which sets all of the text widgets to the new function's comment information and program text.

Toggled
CheckBoxCallback

These set the values for the Comment Style and Include Information (respectively) from the Save Info popup widget.

FileModified

This is a very important callback, for without this, the user would either not be able to modify the program text, or if he did, if would really screw things up. What this function does is calculate the number of positions that the user has inserted or deleted *every time* a modification is made to the program text. After it gets this information, it figures out exactly what function the user has modified and then adjusts the starting positions of all functions which follow it in the file. This ensures that when the user selects another file from the function list, he will still get the correct starting position; also more importantly, this ensures that when saving the file, everything will be saved properly.

QuitCallback
CancelCallback
Exit

These callbacks allow the user to leave the program in some way. With QuitCallback, a small dialog is first popped up asking the user if he really wants to quit. Exit quits altogether; CancelCallback is the callback for the cancel button on the Save Info Window, and just destroys that dialog box.

3.3.3 The Save Info Window

SaveCallback
DoCommentBox
Save
CancelCallback

SaveCallback doubles as the callback for the "Save To File" pushbutton of the Main Window, as well as its own popup widget. Save is the callback for the "Do It" pushbutton of the Save Info Window, and it proceeds to call other procedures such as DoCommentBox and writeheader (in *xcommentBase.c*) to accomplish this task.

4 Conclusion

XComment is a useful aid to programmers at documentation time. It provides an easy-to-use interface along with accurate saving and retrieving of C, C++, and Pascal files. It took *autoc* and enhanced it with a nice-looking graphical user interface which gives programmers more control over how their program is documented. It is no longer an *auto*commenter, it is an interactive documentation tool. XComment could be a useful tool for anybody in the Computer Science Department here at Brown to use, from cs4 students to graduate students to faculty.

Appendix:

The Code

xcomment.h
xcommentFile.c
xcommentBase.c
xcommentUI.c

xcomment.h

```
/*****************************************************************************/
/* Program:   xcomment.h                                                     */
/*                                                                           */
/*                                                                           */
/* XComment by Chris Wood, 1992 (with some code from "autoc", written by     */
/* David Fedor, 1990?)                                                       */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "/cs/src/field/include/FIELD.h"
#include "/cs/src/bwe/auxd/src/auxd.h"
#include "/cs/src/field/include/datatypes.h"
#include <stddef.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

#include <ctype.h>
#include <Xm/List.h>
#include <Xm/LabelG.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/PanedW.h>
#include <Xm/Text.h>
#include <Xm/Frame.h>
#include <Xm/PushB.h>
#include <Xm/ToggleB.h>
#include <Xm/DialogS.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/PushBG.h>


#define LINELEN 512     /* maximum line length read in from files. */

#ifndef XREF_CMD
#define XREF_CMD "/pro/bin/field/xrefserver"
#endif

#define C       0
#define CPP     1
#define PASCAL  2
#define INCLUDE 3
#define TAB     9
#define RETURN 10
#define BLANK  32

extern void initcmtblk();       /* inits variable; call once at start of prog*/
extern void doswitch();         /* parse and perform switch action */
extern void dofile();           /* do autocommenting stuff to file */
extern void AUTOCinit();        /* do stuff for use in meadow */

int totalFunctions;          /* total number of functions in filé */
int startLine[100];          /* holds starting lines for each function */
int startFuncPos[100];       /* holds starting function positions */
int thingType[100];          /* holds type of function/procedure/prog. */
char *programText;           /* holds entire program text */
char *explanationText[100];  /* holds explanation text */
char *functionName[100];     /* holds function names */
char *calledBy[100];         /* holds functions which call each func. */
char *calls[100];            /* holds functions which each func. calls */
int curItem;                 /* current function number */
char *fileName;              /* filename of file */
unsigned long commentInfo;   /* holds bitwise rep. of what is included */
Widget fileRC, message;      /* widgets used for popup message */
int fileType;                /* type of file, either C, CPP, or PASCAL */
char *beginComment,          /* begin comment */
     *endComment;            /* end comment */
char *shortFn;               /* filename without the full path */
Boolean boxComments;         /* true if the user selects box comments */
Boolean commandLine;         /* true if user typed filename on command line */
char *oldFn;
```

xcommentFile.c

```
/*****************************************************************************/
/* Program:   xcommentFile.c                                                 */
/*                                                                           */
/*                                                                           */
/* Author:  Chris Wood                                                       */
/* Date:    May 1, 1992                                                      */
/*                                                                           */
/* This module contains the toplevel user interface for the xautoc, which is */
/* only called if the user does not supply an argument on the command line.  */
/* It also                                                                   */
/* includes the file selection box, which displays only those files which    */
/* are legal: either .c or .C or .p files, depending on which kind of file   */
/* the user has chosen to open.  It then calls DoFile to get things going,   */
/* which is located in xcommentbase.c.                                       */
/*                                                                           */
/* (By the way, this file was commented using xcomment.)                     */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/



#include "xcomment.h"


Boolean openActivated, selectionBoxUp;
int fileArgc;
char *fileArgv;
Widget fileWidget, openButton1, openButton2,
       openButton3, openButton4, quitButton, selectionBox;
XtAppContext app;




/*****************************************************************************/
/* Function:  GetDefaults                                                    */
/*                                                                           */
/* Called By: Open, main                                                     */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Sets up the pertinent information, depending on which kind of file the    */
/* user has chosen to open.  The variables which are set are global and      */
/* defined in xcomment.h, since they are used many times throughout the      */
/* application.                                                              */
/*****************************************************************************/


XmString
GetDefaults (type)
int type;
{
    XmString filePattern;

    beginComment = (char *)MALLOC(3);
    endComment = (char *)MALLOC(3);

    if (type==C) {
        filePattern = XmStringCreateSimple ("*.c");
        fileType = C;
        beginComment = "/*";
        endComment = "*/";
    }
    else if (type==CPP) {
        filePattern=XmStringCreateSimple ("*.C");
        fileType = CPP;
        beginComment = "//";
        endComment[0] = 0;
    }
    else if (type==PASCAL) {
        filePattern=XmStringCreateSimple ("*.p");
        fileType = PASCAL;
        beginComment = "{";
        endComment = "}";
    }
    return (filePattern);
}



/*****************************************************************************/
/* Function:  ReadFile                                                       */
/*                                                                           */
/* Called By: main, open_file                                                */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Reads the filename, gets the "short filename" for the list widget (this   */
/* is only needed if the user has used the File Selection Box).  If we're    */
/* opening a C++ file, we have to do a little sneaky work since we want the  */
/* scanner to think it's a C file (since a scanning of a C++ file for some   */
/* reason produces catastrophic results).  Therefore, we create a temporary  */
/* file called "xcomment.c", and that is what's run through the scanner.     */
/* The user, however, has no idea this is happening, since all of the        */
/* strings bearing the filename will remain the string he typed in or        */
/* selected.  Finally, at the end, the temp file will be removed.            */
/*****************************************************************************/


void ReadFile ()
{
    struct stat statb;
    FILE *fp;
    FILE *inFile, *outFile;
    char line[LINELEN];
    int i;
    char *newFn;
    Boolean found;
    char buf[BUFSIZ];

    if (!commandLine) {
        found=FALSE;
        i=strlen(fileName)-1;
        while ((!found) && (i>=0)) {
            if (fileName[i]=='/') found=TRUE;
            else i--;
        }
    shortFn = &fileName[i+1];
    }

    else {
        shortFn = MALLOC (strlen(fileName)+1);
        strcpy (shortFn, fileName);
```

xcommentUI.c

```
    if (debugging) fprintf(dbg,"Reloaded %s\n",infn);
    if (cmtlines[0].generate)   /* if we should make block comments for them */
        linefp = fctlinefile(infn);       /* file holds line #s of fct decls */
    else linefp = NULL;
    if (debugging && cmtlines[0].generate) fprintf(dbg,"Forcing comments.\n");

    docontent(infn,infp,linefp,1);        /* do the work on the file */
    if (debugging) fprintf(dbg,"Translation complete for %s\n",infn);
    fclose(infp);
    if (!toStdout)
        fclose(outfp);
    if (linefp != NULL) {
        fclose(linefp);
        unlink(linefile);
    }

    if ((!exdoc) && (!toStdout)) {
        backfn=(char *) MALLOC(strlen(infn)+4); /* add 3 chars + null */
        strcpy(backfn,infn);
        strcat(backfn,"~ac");
        unlink(backfn);         /* kill the backup file if it exists */
        if (!rename(infn,backfn))
            if (rename(outfn,infn)) {   /* move the original to the backup */
                fprintf(stderr,"AUTOC: Can't update %s\n",infn);
                if (debugging) fprintf(dbg,"df can't update %s here.",infn);
                return;
            }
            else ;
        else {
            fprintf(stderr,"AUTOC: Can't change %s\n",infn);
            if (debugging) fprintf(dbg,"df can't change %s here.",infn);
            return;
        }
    }
    if (debugging) {
        fclose(dbg);
        if (killdbgfile)        /* we don't want it if we completed normally */
            unlink("/tmp/autoc_debug");
    }
}
```

```
/*****************************************************************************/
/* Function:  initcmtblk                                                     */
/*                                                                           */
/* Called By: DoFile                                                         */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Inits variable; call once at start of program                             */
/*****************************************************************************/

void initcmtblk()
{
  cmtblock[0]=NULL;     /* initialize array which holds comment lines */
                        /* while a block header is being read in      */
}
```

```
/*****************************************************************************/
/* Function:  DoFile                                                         */
/*                                                                           */
/* Called By: (none)                                                         */
/* Calls:     dofile, initcmtblk                                             */
/*                                                                           */
/* Called by either main or open_file (in module "xcommentFile.c").  Sets up */
/* memory for certain text that we may find in the file.  Then calls dofile  */
/* which gets things going.                                                  */
/*****************************************************************************/

void DoFile (argc,argv)
int argc;
char *argv[];
{
    int index;
    int i;
    char *p;

    if (commandLine) printf ("Scanning File...\n");
    argc1 = argc;
    argv1 = argv;
    totalFunctions = 0;

    for (i=0; i<100; i++) {
        functionName[i] = MALLOC(30);
        calls[i] = MALLOC(80);
        calledBy[i] = MALLOC(80);
        explanationText[i] = MALLOC(1000);
    }

    initcmtblk(); /* init the variable */

    MSGinit(0); /* 1 to turn on debugging. */
    MSGservice_start("XREF", NULL, commandName, commandArguments);

    cmtlines[0].generate = 1;           /* turn on generation */
    cmtlines[1].generate = 1;
    cmtlines[2].generate = 1;
    toStdout=1;

    if (fileName[strlen(fileName)-2] == '.')
      dofile(fileName,0);
    else {
        printf("Sorry, can't run on executables yet.\n");
        /* What we want to do here is run a query telling us what */
        /* the files are, then call dofile() on each of them.     */
    }
}
```

```
            }
            else {      /* comment is after declaration.  Dump dec, go on. */
                if (putcode)
                    fputs(temp,outfp);   /* put the declaration line out */
            }
            lno++;      /* this is for the fgets before the if. */
            if ((thingtype==2) && (totalFunctions!=1)) totalFunctions--;
        }
        if ((*line != opencmt) && putcode)
            fputs(line,outfp);
        else if ((!strncmp(line+1,"*Autoc",6))
                 || (!strncmp(line+1,"*autoc",6)) || goodheader(line)) {
            if (debugging) fprintf(dbg,"Fixing cmt at line %d\n",lno);
            startlno = lno;

            getfctinfo(infn,startlno,lno,fctname,&thingtype,totalFunctions-1);
            if (*fctname == '\0') {
                strcpy(fctname,"_NOT_FOUND_");
                if (debugging)
                    fprintf(dbg,"No fct found lines %d-%d\n",startlno,lno);
            }
            generatecmtline= nextfctline(linefp);
        }
        else if (putcode) fputs(line,outfp);
    }

    if (stat (fileName, &statb) == -1 ||
        (statb.st_mode & S_IFMT) != S_IFREG ||
        !(fp = fopen (fileName, "r"))) {
        sprintf (buf, "Can't read %s.", fileName);
    }

    len = statb.st_size;
    programText = MALLOC (len+1000);
    programText = &infn;

    if (!(programText = XtMalloc ((unsigned)(len+1)))) /* +1 for NULL */
        sprintf (buf, "%s: XtMalloc(%ld) failed", len, fileName);
    else {
        if (fread (programText, sizeof (char), len, fp) != len)
            sprintf (buf, "Warning: did not read entire file!");
        else
            sprintf (buf, "Loaded %ld bytes from %s.",len,fileName);
        programText[len] = 0; /* NULL-terminate */
    }

    AdjustStartingPositions ();

    ExtractComments ();

    StartUI(argc1,argv1);
}



/*****************************************************************************/
/* Function:  dofile                                                         */
/*                                                                           */
/* Called By: DoFile, handleMsg                                              */
/* Calls:     docontent, fctlinefile                                         */
/*                                                                           */
/* Load the file and pass it to docontent to get the starting positions of   */
/* the functions/procedures.                                                 */
/*****************************************************************************/

void dofile(infn,exdoc)
char *infn;
int exdoc;      /* true if we're generating exdoc. */
{
    FILE *outfp, *infp, *linefp;
    char *outfn, *backfn;

    if (!commandLine) UpdateMessage (message);

    fileName = infn;

    /* NOTE: Although the following line looks useless, for some reason
       if you comment it out or delete it, the program will seg fault
       during scanning!!!! So don't get rid of this unless you figure
       out why it does that. -caw */

    printf ("                        \n");

    if (infn[strlen(infn)-1] == 'p') {
        opencmt='{';
        closecmt='}';
    }
    else {
        opencmt='/';
        closecmt='/';
    }

    if (debugging) {
        /*dbg = stderr; */
        dbg = fopen("autoc_debug", "w");
    }

    if (toStdout)
        outfp = stdout;
    else {
        outfn=(char *) MALLOC(strlen(infn)+7); /* add 6 chars +null */
        strcpy(outfn,infn);
        if (exdoc) {
            strcat(outfn,".exdoc");
            unlink(outfn);      /* kill the old exdoc file if it exists */
            outfp = fopen(outfn,"w");
        }
        else {
            strcat(outfn,"XXXXXX");
            mktemp(outfn);
            outfp = fopen(outfn,"w");
        }
    }

    if (outfp == 0) {
        fprintf(stderr,"AUTOC: df Can't create output file\n");
        if (debugging) fprintf(dbg,"df can't create tf here.");
        return;
    }

    infp = fopen(infn,"r");
    if (infp == 0) {
        fprintf(stderr,"AUTOC: df Can't open %s\n",infn);
        if (debugging) fprintf(dbg,"df can't open %s here.",infn);
        return;
    }
    MSGcalla("XREF RELOAD %s",infn);       /* tell xref to reload the source */
```

```
  for (i=0; i<totalFunctions; i++) {

      for (j=curLoc; curLine<(startLine[i]-1); j++)
          if (programText[j]==RETURN) curLine++;

      if (fileType!=PASCAL) {
          t=j;
          done = FALSE;
          moveJ = FALSE;
          parenFound = FALSE;
          while (programText[t]!=RETURN) {
              if (programText[t]=='(') parenFound = TRUE;
              t++;
          }
          if (!parenFound) {
              cmtLines=2;
              done=TRUE;
              t++;
          }

          t=j;
          if (!done) {
              while ((programText[t]!=BLANK) && (programText[t]!='('))
                  t++;
              if (programText[t]=='(') moveJ = TRUE;
              else {
                  t++;
                  while (programText[t]==BLANK) t++;
                  if (programText[t]!='(') cmtLines=1;
                  else moveJ = TRUE;
              }
          }
          if (moveJ) {
              cmtLines=2;
              j-=2;
              while (programText[j]!=RETURN) j--;
              startLine[i]--;
              curLine--;
          }
          if (programText[j-1]!=RETURN) j--;
      }

      if (fileType!=C) {
          c = strlen (beginComment);
          while ((strncmp (&programText[j],beginComment,c)==0) ||
                 ((fileType==PASCAL) &&
                  (strncmp (&programText[j],endComment,c)==0))) {
              while (programText[j]!=RETURN) j++;
              j++;
              curLine++;
              startLine[i]++;
          }
          while (programText[j]==RETURN) {
              j++;
              curLine++;
              startLine[i]++;
          }
      }

      if (i==0) startFuncPos[0]=0;
      startFuncPos[i] = j;
      curLoc = j;
  }
}
```

```
/*****************************************************************************/
/* Function:  docontent                                                      */
/*                                                                           */
/* Called By: dofile                                                         */
/* Calls:     nextfctline                                                    */
/*                                                                           */
/* Do the work on the file.  This was originally the heart of autoc.         */
/*****************************************************************************/

void docontent (infn,infp,linefp,putcode)
char *infn;                     /* input file name */
FILE *infp, *linefp;            /* input file, linenumber file. */
int putcode;                    /* true if should put code to outfp */
{
    int index, thingtype;
    int lno = 0;
    int startlno;       /* holds line number of the start of the comment */
    int generatecmtline;
    int i;
    FILE *outfp;
    char line[LINELEN];
    char temp[LINELEN];
    char fctname[LINELEN];
    char buf[LINELEN];
    struct stat statb;
    long len;
    FILE *fp;
    int curLine,curLoc,j,k,t;
    char firstline[LINELEN];
    Boolean done,blankFound,parenFound, moveJ;
    int cmtLines;

    generatecmtline= nextfctline(linefp);   /* get lno of next generated cmt */
    while (fgets(line,LINELEN,infp) != 0) {
        lno++;
        currentFilePos += strlen(line);
        if (lno == generatecmtline) {
            startFuncPos[totalFunctions] = currentFilePos;
            startLine[totalFunctions] = lno;
            strcpy(temp,line);  /* keep the declaration line till later */
            fgets(line,LINELEN,infp);
            if (!( (!strncmp(line+1,"*Autoc",6))||(!strncmp(line+1,"*autoc",6))
                                                || goodheader(line) )){
                if (debugging) fprintf(dbg,"Generating at line %d\n",lno);
                getfctinfo(infn,lno,lno,fctname,&thingtype,totalFunctions);
                thingType[totalFunctions] = thingtype;
                totalFunctions++;

                if ((cmtabove == 0) && putcode)
                    fputs(temp,outfp);   /* put the declaration line out */

                if (thingtype != 2)
                    getgeneratedlines(infn,fctname,outfp);
                else fputs("\n",outfp);

                if ((cmtabove == 1) && putcode)
                    fputs(temp,outfp);  /* put the declaration line out */
                generatecmtline= nextfctline(linefp);
```

```
            while ((strncmp (&tempBuf[curPos], beginComment, c1)!=0)
               && ((tempBuf[curPos]==BLANK) || (tempBuf[curPos]==RETURN)));
                curPos++;

            if (cmtBegPos==-1) cmtBegPos = curPos;

            if ((strncmp (&tempBuf[curPos], beginComment, c1)==0)) {
                cmtFound = TRUE;
                if (fileType==CPP) while (tempBuf[curPos]!=RETURN)
                    curPos++;
                else while (strncmp (&tempBuf[curPos],endComment, c2)!=0)
                    curPos++;
                cmtEndPos = curPos+c2;
                curPos+=c2;
            }
            else done = TRUE;
        }
        /*printf ("startingPos = %d; beg = %d; end = %d\n",
                startingPos,cmtBegPos,cmtEndPos);*/
        if (cmtFound) {
            /*printf ("Comment found in program header...\n");
            printf ("startingPos = %d; beg = %d; end = %d\n",
                    startingPos,cmtBegPos,cmtEndPos);*/
            GetComment (startingPos, cmtBegPos, cmtEndPos, tempBuf, -1);
        }
        else explanationText[i] = "\0";
    }

    startingPos = startFuncPos[i];
    totSize = startFuncPos[i+1]-startingPos;
    tempBuf = MALLOC (totSize+1);
    strncpy (tempBuf, &programText[startingPos], totSize);
    /*if (i<6) printf ("tempBuf = %s\n(end)\n", tempBuf);*/
    tempBufSize = strlen (tempBuf);
    curPos = tempBufSize-1;
    cmtFound = FALSE;
    done = FALSE;
    cmtEndPos = -1;
    cmtBegPos = 0;
    lastReturn = curPos;
    while (!done) {
        if (fileType==CPP) {
            curPos++;
            while (tempBuf[curPos]!=RETURN) curPos--;
        }

        else while ((strncmp (&tempBuf[curPos], endComment, c2)!=0) &&
                   ((tempBuf[curPos]==BLANK) || (tempBuf[curPos]==RETURN)))
            curPos--;

        prevReturn = lastReturn;
        if (fileType==CPP) lastReturn = curPos+1;
        if ((fileType==CPP) && (cmtEndPos==-1)) firstReturn = curPos+1;

        if (cmtEndPos==-1) cmtEndPos = curPos;

        if (fileType!=PASCAL) curPos--;
        if ((fileType==CPP) ||
            ((strncmp (&tempBuf[curPos], endComment, c2)==0))) {
            cmtFound = TRUE;
            chars = FALSE;
            while ((strncmp (&tempBuf[curPos],beginComment,c1)!=0)
                && (!done)) {
                if ((fileType==CPP) && (tempBuf[curPos]==RETURN) &&
                    (chars)) {
                    done = TRUE;
                    curPos = lastReturn+1;
                    if (lastReturn==firstReturn) cmtFound = FALSE;
                }
                if ((fileType==CPP) && (tempBuf[curPos]!=BLANK) &&
                    (tempBuf[curPos]!=RETURN))
                    chars = TRUE;
                curPos--;
            }
            cmtBegPos = curPos;
            curPos--;
        }
        else done = TRUE;
    }

    if (cmtFound) {
        /*printf ("Comment found in function %s...\n",functionName[i]);
        printf ("startingPos = %d; beg = %d; end = %d\n",
                startingPos,cmtBegPos,cmtEndPos);*/
        GetComment (startingPos, cmtBegPos, cmtEndPos, tempBuf, i);
    }
    else {
        /*printf ("No comments found in function %s\n",functionName[i]);*/
        explanationText[i+1] = "\0";
    }

}

fclose (filePtr);
}
```

```
/*****************************************************************************/
/* Function:  AdjustStartingPositions                                        */
/*                                                                           */
/* Called By: docontent                                                      */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Sometimes, the scanner doesn't give us the exact position that we need in */
/* order to search for comments.  This function is called after the scanner  */
/* has done its work, to make sure we have what we want.  Usually, if we     */
/* have a declaration which is two lines long, we must back up to the        */
/* beginning of the first line.  Also, sometimes for Pascal files, the       */
/* scanner gives us the position a couple of lines above the actual          */
/* declaration, which sometimes puts us in the middle of a comment, which    */
/* screws things up.  It sometimes does the same for C++ files.  This       */
/* function accounts for *most* situations.  Sometimes the scanner may       */
/* return a totally bizarre position which this may not be able to fix.  In  */
/* such cases (which are rare), the user might be somewhat screwed.  But     */
/* usually, this does the trick.                                             */
/*****************************************************************************/

void AdjustStartingPositions ()
{
    Boolean done, parenFound, moveJ;
    int i,j,t,c;
    int curLine, curLoc, cmtLines;

    curLine = 0;
    curLoc = 0;
    startFuncPos[0]=0;
```

```
    numCmtLines = 0;

    while (j<strlen(comment)) {

        if ((fileType!=CPP) && (strncmp (&comment[j],endComment,c2)==0)) {
            k=j-1;
            while (comment[k]==BLANK) k--;
            strcpy (&comment[k+1], &comment[j+c2]);
            j=k+1;
        }

        else if (strncmp (&comment[j],beginComment,c1)==0) {
            if (comment[j+c1]==BLANK) num=1;
            else num=0;
            strcpy (&comment[j], &comment[j+c1+num]);
        }
        else if (strncmp (&comment[j],"* * ", 3)==0)
            strcpy (&comment[j], &comment[j+3]);

        else if (strncmp (&comment[j],"**",1)==0)
            strcpy (&comment[j], &comment[j+1]);

        else if (comment[j]==RETURN) {
            numCmtLines++;
            j++;
        }
        else j++;
    }

    for (j=i+1; j<totalFunctions; j++) startLine[j] -= numCmtLines;

    StripOutRedundantInfo (comment, i);
}
```

```
/*****************************************************************************/
/* Function:  GetComment                                                     */
/*                                                                           */
/* Called By: ExtractComments                                                */
/* Calls:     StripOutCommentChars                                           */
/*                                                                           */
/* Given a starting position and and ending position of the comment, this    */
/* function extracts the comment from the program text, and concatenates the */
/* program text so that the comment is no longer there.  Then it calls       */
/* StripOutCommentChars, which will do the work on the comment to get it     */
/* ready for the Explanation Text widget.                                    */
/*****************************************************************************/

void GetComment (startingPos, cmtBegPos, cmtEndPos, tempBuf, i)
int startingPos;
int cmtBegPos;
int cmtEndPos;
char *tempBuf;
int i;
{
    char *comment;
    int k, cmtSize;

    cmtSize = cmtEndPos-cmtBegPos+1;
    comment = MALLOC (cmtSize+1);
    strncpy (comment, &tempBuf[cmtBegPos], cmtSize);
    strcpy (&programText[cmtBegPos+startingPos],
            &programText[cmtEndPos+startingPos+1]);
    for (k=i+1; k<totalFunctions; k++)
        if (k>0) startFuncPos[k] -= cmtSize;
    StripOutCommentChars (comment, i);
}
```

```
/*****************************************************************************/
/* Function:  ExtractComments                                                */
/*                                                                           */
/* Called By: docontent                                                      */
/* Calls:     GetComment                                                     */
/*                                                                           */
/* ExtractComments figures out the beginning and ending positions of header  */
/* comments for each function.  It copies the portion of the program text    */
/* from a starting line to one line less than the next function's starting   */
/* line, then searches from the bottom of that buffer up, until it either    */
/* finds a bunch of comments or a foreign character outside of any comment,  */
/* which is assumed to be the end of the previous function.  If a comment is */
/* found, it sends all pertinent information to GetComment, which begins to  */
/* prepare the comment for insertion into the function explanation widget.   */
/*****************************************************************************/

void ExtractComments ()
{
    char *tempBuf;
    int i,j,k;
    FILE *filePtr, *cmtFile;
    int tempBufSize, curPos;
    int cmtBegPos, cmtEndPos;
    Boolean done, cmtFound, chars;
    int startingPos, totSize;
    int cmtSize, c1, c2;
    int lastReturn, firstReturn, prevReturn;

    filePtr = fopen ("xcomment.tmp","r");

    c1 = strlen (beginComment);
    c2 = strlen (endComment);
    startFuncPos[0] = 0;
    for (i=0; i<totalFunctions-1; i++) {
        cmtBegPos = -1;
        cmtEndPos = -1;
        if (i==0) startingPos=0;
        else startingPos = startFuncPos[i];
        totSize = startFuncPos[i+1]-startingPos;
        tempBuf = MALLOC (totSize+1);
        strncpy (tempBuf, &programText[startingPos], totSize);
        tempBufSize = strlen (tempBuf);
        /*printf ("tempBufSize = %d\n",tempBufSize);*/

        done = FALSE;
        cmtFound = FALSE;
        if (fileType==CPP) c2=1;

        if (i==0) {
            curPos = 0;
            while (!done) {
```

```
    fputs(fctname,outfp);
    if (boxComments) {
        nameLen = strlen (fctname);
        for (index=1; index<colwidth - (nameLen + 16); index++)
            fputs(" ",outfp);
        fputs (endComment,outfp);
    }
    fputs ("\n",outfp);
    addblankcommentline (outfp);
}
```

```
/*****************************************************************************/
/* Function:  goodheader                                                     */
/*                                                                           */
/* Called By: docontent                                                      */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Returns true if a header has been found.                                  */
/*****************************************************************************/

int goodheader(line)
char *line;
{
    char star1[LINELEN];        /* holds stars from input line */
    char star2[LINELEN];
    char fctname[LINELEN];      /* name of the function */
    char fctproc[LINELEN];      /* holds "Function" or "Procedure" */

   if ((*line == opencmt) && (*(line+1) == '*'))
        if (4==sscanf(line+1,"%[*] %s%s %[*]\n",star1,fctproc,fctname,star2)){
            if ((streq(fctproc,"Function") || streq(fctproc,"Procedure")
                 || streq(fctproc,"Program")) &&
                (strlen(star1) ==
                        strlen(star2)+(strlen(fctproc)+strlen(fctname))%2 ))
                return 1;
        }
    return 0;
}
```

```
/*****************************************************************************/
/* Function:  StripOutRedundantInfo                                          */
/*                                                                           */
/* Called By: StripOutCommentChars                                           */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Iterates through the comment that is passed to it, looking for any of the */
/* keywords that might identify a line that we don't need in the explanation */
/* text, since the information will be contained in other widgets.  The      */
/* exact form of each of these words is the same that XComment saves them    */
/* as; therefore, if the user has used XComment to comment their program in  */
/* the past, it should work flawlessly.  Otherwise, a slight misspelling or  */
/* format difference may not be caught and the line will not be pulled from  */
/* the explanation text.  (This is no big deal; the user can just delete it  */
/* manually.)  It is assumed that the whole line is unnecessary, so the      */
/* entire line is stripped out and trashed.                                  */
/*****************************************************************************/
```

```
void StripOutRedundantInfo (comment, i)
char *comment;
int i;
{
    int j,k;

    for (j=0; j<strlen(comment); j++) {
        if ((strncmp (&comment[j], "Called By:", 10) == 0) ||
            (strncmp (&comment[j], "Calls:", 6) == 0) ||
            (strncmp (&comment[j], "Function:", 9) == 0) ||
            (strncmp (&comment[j], "Procedure:", 10) == 0) ||
            (strncmp (&comment[j], "Program:", 8) == 0))
            {
                k=j;
                while (comment[k]!=RETURN) k++;
                if (k+1<strlen(comment)) strcpy (&comment[j], &comment[k+1]);
                else comment[j]=0;
                j--;
            }
    }

    /* now get rid of leading returns & blanks... */

    j=0;
    while ((comment[j]==RETURN) || (comment[j]==BLANK)) j++;
    strcpy (&comment[0], &comment[j]);

    /* stick comment into explanationText widget... */

    /*printf ("comment:\n%s\n",comment);*/
    explanationText[i+1] = (char *)MALLOC (strlen (comment) + 1000);
    if (strlen(comment)==0) explanationText[i+1]="\0";
    else strcpy (explanationText[i+1], comment);
}
```

```
/*****************************************************************************/
/* Function:  StripOutCommentChars                                           */
/*                                                                           */
/* Called By: GetComment                                                     */
/* Calls:     StripOutRedundantInfo                                          */
/*                                                                           */
/* This function takes a comment and strips out all of its begin and end     */
/* comments.  For C files, this means the strings "/<star>", "<star>/",      */
/* and any single stars (just to make sure).  Pascal files will be stripped  */
/* of ")", "{", and any single stars,                                        */
/* and C++ files will be stripped of any double slash strings, as well as    */
/* any stars (which under normal circumstances aren't there anyway).  This   */
/* process is done so that the function explanation in the widget will be    */
/* just the comments, and the begin and end comments will be inserted later  */
/* according to the style that the user selects.                             */
/*****************************************************************************/

void StripOutCommentChars (comment, i)
char *comment;
int i;
{
    int j,k,c1,c2,num,numCmtLines;

    c1 = strlen(beginComment);
    c2 = strlen(endComment);
    j=0;
```

```
/*****************************************************************************/
/* Function:  getfctinfo                                                     */
/*                                                                           */
/* Called By: docontent                                                      */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Find the function declared between the given lines.                       */
/*****************************************************************************/

void getfctinfo(infn,startlno,lno,fctname,thingtype,num)
char *infn;
int startlno,lno;
char *fctname;
int *thingtype;
int num;
{
    char *replyfile, *query;
    char data[LINELEN];
    char dtype[LINELEN];
    FILE *datafp;
    int i, newPos;

    if (startlno < 3)   /* make sure we don't go negative on the query! */
        startlno=3;
    query = MALLOC(150+(2*strlen(infn)));
    sprintf(query,"XREF QUERY %s D.name,D.type D.file='%s & D.line>'%d & D.line<'%d & (
D.class='EFUNCTION | D.class='SFUNCTION | D.class='PROGRAM)",infn,infn,startlno-3,lno+1
);
    replyfile = MSGcall(query);
    if (replyfile == NULL || *replyfile == 0 || STREQL(replyfile,"*")) {
        fprintf(stderr,"AUTOC: gfi XREF can't make temp file... Bye!\n");
        if (debugging) fprintf(dbg,"gfi can't do tf here.");
        exit(1);
    }

    datafp = fopen(replyfile,"r");
    if (datafp == 0) {
        fprintf(stderr,"AUTOC: gfi lost data file %s... Bye!\n",replyfile);
        if (debugging) fprintf(dbg,"gfi lost %s here.\n",replyfile);
        exit(1);
    }

    fgets(data,LINELEN,datafp);
    if (!feof(datafp)) {
        sscanf(data,"%[^\177]\177%[^\177]\177",fctname,dtype);

        *thingtype=1;                           /* assume it is a function */
        if (streq(dtype,"void ()"))             /* no, it is a procedure.*/
            *thingtype=0;
        else if (streq(dtype,"*UNKNOWN*"))      /* no, it is a program. */
            *thingtype=2;
    }
    else {      /* no function header was found close by */
        *fctname = '\0';
        *thingtype=0;
    }

    if (thingtype==2) calledBy[0] = "(this is the main program)";

    if ((totalFunctions==0) && (thingtype!=2)) {
        functionName[0] = shortFn;
        calledBy[0] = "(this is the main program)";
        calls[0] = "";
        startLine[1] = startLine[0];
        startLine[0] = 1;
        thingType[0] = 2;
        totalFunctions++;
        num++;
    }
    strcpy (functionName[num], fctname);
    currentFilePos += strlen(fctname);

    fclose(datafp);
    if (unlink(replyfile) != 0) {
        fprintf(stderr,"AUTOC: gfi couldn't trash %s\n",replyfile);
        if (debugging) fprintf(dbg,"gfi can't trash %s here.",replyfile);
        exit(1);
    }
    FREE(query);
}
```

```
/*****************************************************************************/
/* Function:  writeheader                                                    */
/*                                                                           */
/* Called By: (none)                                                         */
/* Calls:     addblankcommentline                                            */
/*                                                                           */
/* Writes the first line, of each header comment.  This line will either be  */
/* a "Function", "Procedure", or "Program" line.  This function is only      */
/* called if the user has set the proper checkbox to "on" in the Save To     */
/* File popup dialog box.                                                    */
/*****************************************************************************/

void writeheader(fctname,funcNum,outfp)
char *fctname;
int funcNum;
FILE *outfp;
{
    int index, numstars, nameLen;

    if ((thingType[funcNum-1]==2) || (funcNum==0)) {
        if ((boxComments) || (fileType==CPP)) fputs (beginComment,outfp);
        else fputs (" *",outfp);
        fputs(" Program:   ",outfp);
    }

    else if ((fileType==PASCAL) && (thingType[funcNum-1]==0)) {
        if (boxComments) fputs("( Procedure: ",outfp);
        else fputs ("  Procedure: ",outfp);
    }

    else if ((fileType==C) || (thingType[funcNum-1]==1)) {
        if ((boxComments) || (fileType==CPP)) fputs (beginComment,outfp);
        else fputs (" *",outfp);
        fputs(" Function:  ",outfp);
    }
    else if (fileType==CPP)
        fputs("// Function:  ",outfp);
```

```
    expText = MALLOC (strlen (explanationText[j-1])+1);
    strcpy (expText, explanationText[j-1]);
    curLine = 0;
    line = 0;
    curLen = 0;
    maxLen = colwidth-6;
    expLen = strlen(expText);
    if (expText[expLen-1]!=RETURN) {
        expText[expLen] = RETURN;
        expText[expLen+1] = "\0";
    }
    expLen = strlen(expText);
    for (i=0; i<expLen; i++) {
        curLen++;
        if (expText[i]==TAB) expText[i]=BLANK;
        if (expText[i]==BLANK) lastBlank = i;
        if ((expText[i]==RETURN) || (curLen>=maxLen)) {
            if (expText[i]!=RETURN) i=lastBlank;
            expText[i]=0;
            expLine[line] = &expText[curLine];
            line++;
            curLine = i+1;
            curLen = 0;
        }
    }

    for (i=0; i<line; i++) {
        if ((boxComments) || (fileType==CPP)) fputs (beginComment,outfile);
        else fputs (" *",outfile);
        fputs (" ",outfile);
        fputs (expLine[i],outfile);
        if (boxComments) {
            expLen = strlen (expLine[i]);
            for (index=1; index<colwidth - (expLen + 5); index++)
                fputs(" ",outfile);
            fputs (endComment,outfile);
        }
        fputs ("\n",outfile);
    }
}
```

```
/*****************************************************************************/
/* Function:  addgeneratedlines                                              */
/*                                                                           */
/* Called By: (none)                                                         */
/* Calls:     addblankcommentline, addexplanationtext                        */
/*                                                                           */
/* Adds the generated lines, which are obtained from the "Called By" and     */
/* "Calls" widgets.  Only adds these lines if the user has turned their      */
/* generating checkboxes on in the Save to File popup dialog box.            */
/*****************************************************************************/

int addgeneratedlines(j,outfile)
int j;
FILE *outfile;
{
    int index,nameLen;

    if (thingType[j-1]!=2) {
        if (2 & commentInfo) {
            if ((boxComments) || (fileType==CPP))
                fputs (beginComment,outfile);
            else fputs (" *",outfile);
            fputs (" Called By: ",outfile);
            fputs (calledBy[j-1],outfile);
            if (boxComments) {
                nameLen = strlen (calledBy[j-1]);
                for (index=1; index<colwidth - (nameLen + 16); index++)
                    fputs(" ",outfile);
                fputs (endComment,outfile);
            }
            fputs ("\n",outfile);
        }

        if (4 & commentInfo) {
            if ((boxComments) || (fileType==CPP))
                fputs (beginComment,outfile);
            else fputs (" *",outfile);
            fputs (" Calls:     ",outfile);
            fputs (calls[j-1],outfile);
            if (boxComments) {
                nameLen = strlen (calls[j-1]);
                for (index=1; index<colwidth - (nameLen + 16); index++)
                    fputs(" ",outfile);
                fputs (endComment,outfile);
            }
            fputs ("\n",outfile);
        }
    }

    if (((strlen (explanationText[j-1]))!=0) && (8 & commentInfo)) {
        addblankcommentline (outfile);
        addexplanationtext (j,outfile);
    }
}
```

```
/*****************************************************************************/
/* Function:  getgeneratedlines                                              */
/*                                                                           */
/* Called By: docontent                                                      */
/* Calls:     putcmtline                                                     */
/*                                                                           */
/* Gets the generated lines from the input file.                             */
/*****************************************************************************/

int getgeneratedlines(infn,fctname,outfp)
FILE *infn;
char *fctname;
FILE *outfp;
{
    int index = 0;

    while (cmtlines[++index].text != NULL)
        if ((cmtlines[index].generate) || (cmtlines[index].required)) {
            if (debugging) fprintf(dbg,"Generating %s\n",cmtlines[index].text);
            putcmtline(infn,fctname,index,outfp);
        }
}
```

```
/*****************************************************************************/
/* Function:  putcmtline                                                     */
/*                                                                           */
/* Called By: getgeneratedlines                                              */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Do the query and put out the given comment                                */
/*****************************************************************************/

void putcmtline(infn,fctname,index,outfp)
char *infn, *fctname;
int index;      /* comment number to do */
FILE *outfp;

{
    char *replyfile;
    FILE *datafp;
    char oline[LINELEN];        /* output line */
    char line[LINELEN];         /* input file line */
    char word[LINELEN];         /* word from input line */
    char query[LINELEN];        /* query string that is sent as a MSG */
    char *olineStr;

    strcpy(oline,cmtlines[index].text);
    sprintf(query,cmtlines[index].query,infn,fctname);

    replyfile = MSGcall(query);
    if (replyfile == NULL || *replyfile == 0 || STREQL(replyfile,"*")) {
        fprintf(stderr,"AUTOC: fl XREF call returned nothing.\n");
        if (debugging) fprintf(dbg,"fl XREF returned 0 here.\n");
        exit(1);
    }
    datafp = fopen(replyfile,"r");
    if (datafp == 0) {
        fprintf(stderr,"AUTOC: fl lost data file %s. Aborting.\n",replyfile);
        if (debugging) fprintf(dbg,"fl lost %s here.",replyfile);
        exit(1);
    }

    fgets(line,LINELEN,datafp);
    if feof(datafp)
        strcat(oline,"(none)");
    while (!feof(datafp)) {
        sscanf(line,"%[^\177]",word);
        if (strlen(oline)+strlen(word)+2 < colwidth-cmtlines[0].minlen)
            strcat(oline,word);
        else {
            fputs(oline,outfp);
            fputs("\n",outfp);
            strcpy(oline,cmtlines[0].text);
            strcat(oline,word);
        }
        fgets(line,colwidth-cmtlines[0].minlen,datafp); /* get next line */
        if (!feof(datafp))      /* if there's more words coming */
            strcat(oline,", ");
    };
    fputs(oline,outfp);
    if (index==1) {
        olineStr = &oline[13];
        strcpy (calledBy[totalFunctions-1],olineStr);}
    else if (index==2) {
        olineStr = &oline[13];
        strcpy (calls[totalFunctions-1],olineStr);}

    fputs("\n",outfp);
    fclose(datafp);
    unlink(replyfile);
}



/*****************************************************************************/
/* Function:  addblankcommentline                                            */
/*                                                                           */
/* Called By: addgeneratedlines, writeheader                                 */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Adds a blank comment line to the current place in the output file.        */
/* Called only when the user is saving the file.                             */
/*****************************************************************************/

void addblankcommentline (outfile)
FILE *outfile;
{
    int index;

    if (fileType==CPP) {
        fputs (beginComment,outfile);
        fputs ("\n",outfile);
    }
    else if (boxComments) {
        fputs (beginComment,outfile);
        for (index=1; index<colwidth-4; index++)
            fputs(" ",outfile);
        fputs (endComment,outfile);
        fputs ("\n",outfile);
    }
    else fputs (" *\n",outfile);
}



/*****************************************************************************/
/* Function:  addexplanationtext                                             */
/*                                                                           */
/* Called By: addgeneratedlines                                              */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Adds the text from the Explanation Text Widget to the output file.  If we */
/* are using box comments, it must calculate the maximum number of           */
/* characters to put on each line and pad the extra space with blank         */
/* characters before closing the comment line with an endComment string.     */
/*****************************************************************************/

void addexplanationtext (j,outfile)
int j;
FILE *outfile;
{
    int i,curLine,line,expLen,index,maxLen,curLen,lastBlank;
    char *expText, *expLine[50];
```

```
/*****************************************************************************/
/* Program:   xcommentBase.c                                                 */
/*                                                                           */
/*                                                                           */
/* XComment by Chris Wood, 1992 (with some code from "autoc", written by     */
/* David Fedor, 1990?)                                                       */
/*                                                                           */
/* All of the scanning of the input file is done in this module.  Any header */
/* comments are extracted from the file and placed in widgets for the user   */
/* to see.  This part of the program is initiated when either open_file or   */
/* main (both in module "xcommentFile.c") calls function DoFile (at the end  */
/* of this file).                                                            */
/*                                                                           */
/* (Incidentally, all comments in this file were made using xcomment.)       */
/*                                                                           */
/*****************************************************************************/

#include "xcomment.h"

char *commandName = XREF_CMD;
char *commandArguments[] = { XREF_CMD, NULL };
int argc1;
char *argv1;
int currentFilePos=0;

typedef struct CMTSTRUCT {
                  char *text;     /* text of query */
                  int generate;   /* if should add this line when generating */
                  int required;   /* if this line is required for all comts  */
                  int usedhere;   /* if this line is in the current block    */
                  int minlen;     /* length of text field required for input */
                  int nextquery;  /* number of next query to run on the data */
                  char *query;    /* text of the XREF query                  */
                      } cmtstruct;

cmtstruct cmtlines[] = {
    { "             ", 0, 0, 0, 13, 0, "" },     /* empty line data */
        /* the generate field here is the one consulted for the whole block */

    { "Called by:   ", 0, 0, 0, 10, 0, "XREF QUERY %s C.from C.call=='%s'" },

    { "Calls:       ", 0, 0, 0, 6, 0,
          "XREF QUERY %s C.call C.from=='%s & F.name=C.call" }
          /* second condition strips out system calls */
};

#define streq(s1,s2)     (! strcmp(s1,s2))
#define Tolower(c)       (isupper(c) ? tolower(c) : c)



#define MAXHELDLINES 100
static char *cmtblock[MAXHELDLINES];     /* holds one comment block */

char opencmt, closecmt; /* gets set according to which language is used */
int doabox=0;           /* whether to use box style in comments */
extern int colwidth=80; /* column width for comment justification */
int debugging=0;        /* turn on (through messages or switch) to debug */
int formfeed=0;         /* 1= put a ff before procedures */
int killdbgfile=1;      /* 1= remove the debugging file on normal terminate */
char *linefile;         /* holds file name of XREF generate-lines file */
int cmtabove=1;         /* by default, put comment above declaration line */
int toStdout=0;         /* 1= send formatted file to stdout, not to file. */
FILE *dbg;              /* debugging output file */
```

```
/*****************************************************************************/
/* Function:  nextfctline                                                    */
/*                                                                           */
/* Called By: docontent                                                      */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Returns line number of next function declaration.                         */
/*****************************************************************************/

int nextfctline(fctlist)
FILE *fctlist;
{
    char line[LINELEN];
    int theline;

    theline=0;                          /* return 0 if there's no data */
    if ((fctlist != NULL) && (!feof(fctlist))) {
        fgets(line,LINELEN,fctlist);
        if (!feof(fctlist))
            sscanf(line,"%d\177",&theline);
    }
    return theline;
}
```

```
/*****************************************************************************/
/* Function:  fctlinefile                                                    */
/*                                                                           */
/* Called By: dofile                                                         */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Opens file which holds line numbers of function declarations.             */
/*****************************************************************************/

FILE *fctlinefile(infn)
char *infn;
{
    char query[LINELEN];
    FILE *result;       /* holds the result temporarily */

    sprintf(query,"XREF QUERY %s F.line F.file='%s",infn,infn);
    linefile = MSGcall(query);
    if ((linefile == NULL) || (*linefile=='\0') || STREQL(linefile,"")) {
        fprintf(stderr,"AUTOC: flf XREF can't make temp file... aborting.\n");
        if (debugging) fprintf(dbg,"flf can't make tf here.\n");
        exit(1);
    }

    result = fopen(linefile,"r");
    if (result == 0) {
        fprintf(stderr,"AUTOC: flf lost data file %s... aborting.\n",linefile);
        if (debugging) fprintf(dbg,"flf lost %s here.\n",linefile);
        exit(1);
    }

    return result;
}
```

xcommentBase.c

```
/*****************************************************************************/
/* Function:  KillandOpen                                                    */
/*                                                                           */
/* Called By: Open                                                           */
/* Calls:     open_file                                                      */
/*                                                                           */
/* "KillandOpen" kills a dialog box and then opens up a selection box.       */
/*****************************************************************************/

void
KillandOpen (dialog, save, cbs)
Widget dialog;
int save;
XmFileSelectionBoxCallbackStruct *cbs;
{
    XtDestroyWidget (selectionBox);
    selectionBoxUp = FALSE;
    open_file (dialog, save, cbs);
}
```

```
    selectionBoxUp = FALSE;
    display = XtDisplay (fileWidget);

    XtSetArg (args[0], XmNdialogType, XmDIALOG_MESSAGE);
    XtSetArg (args[1], XmNmessageString,
            XmStringLtoRCreate ("Scanning File...",XmSTRING_DEFAULT_CHARSET));
    XtSetValues (message, args, 2);

    /* flush event queue */
    XFlush (display);
    while (XtAppPending (app)) {
        XtAppNextEvent (app, &event);
        XtDispatchEvent (&event);
    }
    XmUpdateDisplay(message);
}
```

```
/*****************************************************************************/
/* Function:  open_file                                                      */
/*                                                                           */
/* Called By: KillandOpen, Open                                              */
/* Calls:     ReadFile, UpdateMessage                                        */
/*                                                                           */
/* Callback routine for the open pushbutton of the file selection dialog.    */
/* calls readfile to check on the validity of the file, etc., then sends the */
/* file off to DoFile (in xcommentbase.c) to get scanned.                    */
/*****************************************************************************/

void
open_file (dialog, save, cbs)
Widget dialog;
int save;
XmFileSelectionBoxCallbackStruct *cbs;
{
    XmString label;
    Display *display;


    if (!XmStringGetLtoR (cbs->value, XmSTRING_DEFAULT_CHARSET, &fileName))
        return;  /* internal error */

    if (!*fileName) {
        XtFree (fileName);
        return;  /* nothing typed */
    }

    ReadFile ();

    XtDestroyWidget (dialog);
    selectionBoxUp = FALSE;

    display = XtDisplay (fileWidget);

    message = XmCreateMessageDialog (fileWidget, "message", NULL, 0);

    label = XmStringCreateLtoR ("Scanning File...", XmSTRING_DEFAULT_CHARSET);
```

```
    XtVaSetValues (message, XmNdialogType, XmDIALOG_MESSAGE,
                   XmNmessageString, label, NULL);
    XtManageChild (message);

    UpdateMessage (message);

    DoFile (fileArgc, fileArgv);

    XtDestroyWidget (message);
}
```

```
/*****************************************************************************/
/* Function:  Kill                                                           */
/*                                                                           */
/* Called By: main                                                           */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Exits program.                                                            */
/*****************************************************************************/


void Kill (w, client_data, cbs)
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *cbs;
{
    exit(0);
}
```

```
/*****************************************************************************/
/* Function:  KillSelectionBox                                               */
/*                                                                           */
/* Called By: Open                                                           */
/* Calls:     (none)                                                         */
/*                                                                           */
/* kills selection file box.                                                 */
/*****************************************************************************/


void KillSelectionBox (w, client_data, cbs)
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *cbs;
{
    XtDestroyWidget (selectionBox);
    selectionBoxUp = FALSE;
}
```

```
                                    NULL);
    XmStringFree (openButton2Text);
    XtAddCallback (openButton2, XmNactivateCallback, Open, NULL);

    openButton3Text = XmStringCreateSimple ("Open Pascal File");
    openButton3 = XtVaCreateManagedWidget("openButton",
                                    xmPushButtonWidgetClass, fileRC,
                                    XmNlabelString, openButton3Text,
                                    XmNfontList, font14pt,
                                    XmNheight, 100,
                                    XmNwidth, 200,
                                    NULL);
    XmStringFree (openButton3Text);
    XtAddCallback (openButton3, XmNactivateCallback, Open, NULL);

    quitButtonText = XmStringCreateSimple ("Quit XComment");

    quitButton = XtVaCreateManagedWidget ("quitButton",
                                    xmPushButtonWidgetClass, fileRC,
                                    XmNlabelString, quitButtonText,
                                    XmNrows, 3,
                                    XmNwidth, 200,
                                    XmNfontList, font14pt,
                                    NULL);

    XmStringFree (quitButtonText);
    XtAddCallback (quitButton, XmNactivateCallback, Kill, NULL);

    XtManageChild (fileRC);
    XtRealizeWidget (fileWidget);

    XtAppMainLoop (app);
}
```

```
/*****************************************************************************/
/* Function:  Open                                                           */
/*                                                                           */
/* Called By: main                                                           */
/* Calls:     open_file                                                      */
/*                                                                           */
/* Callback routine for the file type pushbuttons.  Pops up a file selection */
/* dialog box containing only those files with the appropriate extension.    */
/* then sends the selected file off to open_file (callback routine).  The    */
/* file selection box is later destroyed when the main window is popped up.  */
/*****************************************************************************/


void Open (dialog, save, cbs)
Widget dialog;
int save;
XmFileSelectionBoxCallbackStruct *cbs;
{
     void open_file (), KillSelectionBox(), KillandOpen();
     XmString str, title;
     XmString filePattern;
     Position yLoc, topY;

     XtRealizeWidget (fileWidget);
```

```
     if (!selectionBoxUp) {

        selectionBoxUp = TRUE;

        XtVaGetValues (dialog, XmNy, &yLoc, NULL);
        XtVaGetValues (fileWidget, XmNy, &topY, NULL);

        selectionBox = XmCreateFileSelectionDialog (fileRC, "Files", NULL, 0);

        if (dialog==openButton1) filePattern = GetDefaults (C);
        else if (dialog==openButton2) filePattern = GetDefaults (CPP);
        else if (dialog==openButton3) filePattern = GetDefaults (PASCAL);

        XtAddCallback (selectionBox, XmNcancelCallback, KillSelectionBox,NULL);
        XtAddCallback (selectionBox, XmNokCallback, KillandOpen, NULL);
        XtAddCallback (selectionBox, XmNhelpCallback, XtUnmanageChild, NULL);

        str = XmStringCreateSimple ("Open");
        title = XmStringCreateSimple ("Open File");

        XtVaSetValues (selectionBox,
                       XmNokLabelString, str,
                       XmNdialogTitle, title,
                       XmNheight, 200,
                       XmNwidth, 200,
                       NULL);

        XtVaSetValues (selectionBox,
                       XmNy, yLoc+topY+75,
                       XmNpattern, filePattern,
                       NULL);

        XmStringFree (str);
        XmStringFree (title);
        XtManageChild (selectionBox);
     }
}
```

```
/*****************************************************************************/
/* Function:  UpdateMessage                                                  */
/*                                                                           */
/* Called By: open_file                                                      */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Needed so that text is displayed in the message box, which informs the    */
/* user that the selected file is being scanned.  Usually works ok, however  */
/* sometimes it doesn't refresh like it should.  Don't know why...           */
/*****************************************************************************/


void UpdateMessage (message)
Widget message;
{
    Display *display;
    XEvent event;
    Arg args[3];

    XtDestroyWidget (selectionBox);
```

```
    }

    oldFn = MALLOC (strlen(fileName)+1);
    strcpy (oldFn, fileName);

    if (fileType==CPP) {
        newFn = MALLOC (11);
        newFn = "xcomment.c";
        inFile = fopen (fileName,"r");
        outFile = fopen (newFn,"w");
        fgets (line,LINELEN,inFile);
        while (!feof(inFile)) {
            fputs (line, outFile);
            fgets (line, LINELEN, inFile);
        }
        fclose (inFile);
        fclose (outFile);

        strcpy (fileName, newFn);
    }


    /*make sure the file is a regular text file & open it */

    if (stat (fileName, &statb) == -1 ||
        (statb.st_mode & S_IFMT) != S_IFREG ||
        !(fp = fopen (fileName, "r"))) {
        perror (fileName);
        sprintf (buf, "Can't read %s.", fileName);
        XtFree (fileName);
        return;
    }
    fclose (fp);
}
```

```
/*****************************************************************************/
/* Function:  main                                                           */
/*                                                                           */
/* Called By: (none)                                                         */
/* Calls:     GetDefaults, Kill, Open, ReadFile                              */
/*                                                                           */
/* this function is called from the command line.  if an argument (filename) */
/* is supplied, that file will be checked for validity and opened, and sent  */
/* to DoFile (in xcommentbase.c) to be scanned.  Otherwise, the user will be */
/* provided four pushbuttons (one for C files, one for C++, one for Pascal   */
/* Files, and one to quit XComment).                                         */
/*****************************************************************************/


void main (argc, argv)
int argc;
char *argv[];
{
    Widget dialog;
    XmString str, title;
    XmString openButton1Text, openButton2Text,
             openButton3Text, quitButtonText;
    XmFontList font14pt;
    XFontStruct *font;
```

```
Display *dpy;
void Open(), Kill();
Arg args[3];
FILE *inFile;

fileArgc = argc;
fileArgv = argv;
if (argc>1) {
    commandLine = TRUE;
    fileName = argv[1];
    inFile = fopen (fileName,"r");
    if (inFile == 0) {
        printf("XComment: Can't open %s; aborting...\n",fileName);
        exit (0);
    }
    if (fileName[strlen(fileName)-1] == 'p') GetDefaults (PASCAL);
    else if (fileName[strlen(fileName)-1] == 'c') GetDefaults (C);
    else if (fileName[strlen(fileName)-1] == 'C') GetDefaults (CPP);
    else {
        printf ("Unacceptable file extension; aborting...\n");
        exit (0);
    }
    ReadFile ();
    DoFile (fileArgc, fileArgv);
}
commandLine = FALSE;

selectionBoxUp = FALSE;

fileWidget = XtVaAppInitialize (&app, "file", NULL, 0, &argc, argv,
                                NULL, NULL);

dpy = XtDisplay (fileWidget);
font = XLoadQueryFont (dpy, "-*-courier-bold-r-*--14-*");
font14pt = XmFontListCreate (font, "charset1");

fileRC = XtVaCreateManagedWidget ("fileRC", xmRowColumnWidgetClass,
                          fileWidget,
                          XmNorientation, XmVERTICAL,
                          XmNheight, 1000,
                          XmNwidth, 1000,
                          NULL);

XtSetArg(args[0], XmNwidth, 200);
XtSetArg(args[1], XmNheight, 200);

openButton1Text = XmStringCreateSimple ("Open C File");
openButton1 = XtVaCreateManagedWidget("openButton",
                                  xmPushButtonWidgetClass, fileRC,
                                  XmNlabelString, openButton1Text,
                                  XmNfontList, font14pt,
                                  XmNheight, 100,
                                  XmNwidth, 200,
                                  NULL);
XmStringFree (openButton1Text);
XtAddCallback (openButton1, XmNactivateCallback, Open, NULL);

openButton2Text = XmStringCreateSimple ("Open C++ File");
openButton2 = XtVaCreateManagedWidget("openButton",
                                  xmPushButtonWidgetClass, fileRC,
                                  XmNlabelString, openButton2Text,
                                  XmNfontList, font14pt,
                                  XmNheight, 100,
                                  XmNwidth, 200,
```

```
/*****************************************************************************/
/* Program:   xcommentUI.c                                                   */
/*                                                                           */
/*                                                                           */
/* Author:  Chris Wood                                                       */
/* Date:    May 1, 1992                                                      */
/*                                                                           */
/* This module takes care of the widget which is the main part of the        */
/* program.  It has widgets for the function name, the called by and calls   */
/* parts, as well as the function explanation and a larger text widget for   */
/* the program itself, which displays the program at the location of the     */
/* current function or procedure.                                            */
/*                                                                           */
/* The "toplevel" widget is a popup widget if the user has gone through the  */
/* File Selection Box; otherwise it's the main application widget.  It calls */
/* a popup widget when the "Save To File" button is pressed.  A little menu  */
/* is then popped up with several options that the user can choose between.  */
/*                                                                           */
/* There is also a quit button which reverts the user back to the main menu. */
/*                                                                           */
/* (By the way, all comments in this file were made using the program.)      */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/


#include "xcomment.h"

Widget text_w, calledBy_w, calls_w, funcName_w, exp_text_w, list_w,
       toplevel, dialog, saveButton, filenameText;
XmString label;
XmStringCharSet charset = XmSTRING_DEFAULT_CHARSET;
XmFontList font10pt, font12pt, font14pt, font24pt;
XFontStruct *font;
Display *dpy;
Arg args[8], textArgs[8];
extern int colwidth;
XmString str;
Boolean dialogUp;



/*****************************************************************************/
/* Function:  StartUI                                                        */
/*                                                                           */
/* Called By: (none)                                                         */
/* Calls:     QuitCallback, SaveCallback, selection                          */
/*                                                                           */
/* This is the heart of the user interface.  It sets up the main window,     */
/* with the function list widget along the left side.  It has room for up to */
/* 20 function names; if there are more than 20, the widget will become a    */
/* scrolled list widget.  The upper right part of the window contains the    */
/* Function Name, the Called By and Calls information, and the Function      */
/* Explanation text widget.  Finally, the bottom right part of the window is */
/* the Program Text Widget, which holds the entire program text, displaying  */
/* the top of the present function.                                          */
/*****************************************************************************/



StartUI (argc, argv)
int argc;
char *argv;
{
    XtAppContext app;
    Widget rightRowCol, leftRowCol, functionRowCol, commentInfo_w, label_w,
           topRowCol, lowerRowCol, calledByRowCol, listWidgets, rowcol,
           callsRowCol, expRowCol, leftSide, rightSide, frame, progName,
           bottomRowCol, quitButton, nextButton, prevButton;
    int i,n;
    void selection ();
    XmStringTable str_list;
    XGCValues gcv;
    XmString saveButtonText,quitButtonText,nextButtonText,prevButtonText,
             titleText;
    void QuitCallback(), SaveCallback(), FileModified(),
         NextCallback(), PrevCallback(), Exit();
    int numDisplayed;
    int startingLineNumber,j;
    char *functionText, *title = NULL;
    int startingTextPosition, endingTextPosition;
    XmString defaultItem;
    char *program = NULL;
    char *funcText;
    char *newName[100];
    int same;

    if (fileType==CPP) fileName[strlen(fileName)-1] = 'C';
    dialogUp = FALSE;
    boxComments = TRUE;
    commentInfo = 15;       /* all bits on! */
    n = totalFunctions;
    curItem = 0;

    program = "Program: ";

    if (!commandLine)
        toplevel = XtVaCreatePopupShell ("popup",
                                  xmDialogShellWidgetClass, XtParent(fileRC),
                                  XmNtitle, "xcomment program window",
                                  XmNdeleteResponse, XmDESTROY,
                                  NULL);

    else toplevel = XtVaAppInitialize (&app, "popup", NULL, 0, &argc, argv,
                                       NULL, NULL);

    dpy = XtDisplay (toplevel);
    font = XLoadQueryFont (dpy, "-*-courier-bold-r-*--18-**");
    font24pt = XmFontListCreate (font, "charset1");
    font = XLoadQueryFont (dpy, "-*-courier-bold-r-*--14-**");
    font14pt = XmFontListCreate (font, "charset1");
    font = XLoadQueryFont (dpy, "-*-courier-bold-r-*--12-**");
    font12pt = XmFontListCreate (font, "charset1");
    font = XLoadQueryFont (dpy, "-*-courier-bold-r-*--10-**");
    font10pt = XmFontListCreate (font, "charset1");

    rowcol = XtVaCreateWidget ("rowcol", xmRowColumnWidgetClass,
                               toplevel,
                               XmNorientation, XmVERTICAL,
                               NULL);

    progName = XtVaCreateWidget ("progName", xmPanedWindowWidgetClass,
                                 rowcol,
                                 XmNorientation, XmHORIZONTAL,
                                 NULL);

    frame = XtVaCreateManagedWidget ("frame10",
```

```
                                xmFrameWidgetClass, progName,
                                XmNshadowType, XmSHADOW_OUT,
                                NULL);

    str = XmStringCreateSimple (oldFn);
    label_w = XtVaCreateManagedWidget ("label_w",
                                xmLabelWidgetClass, frame,
                                XmNlabelString, str,
                                XmNfontList, font24pt,
                                NULL);

    XtManageChild (progName);

    bottomRowCol = XtVaCreateWidget ("lowerRowCol", xmRowColumnWidgetClass,
                                rowcol,
                                XmNorientation, XmHORIZONTAL,
                                NULL);

    frame = XtVaCreateManagedWidget ("frame0",
                                xmFrameWidgetClass, bottomRowCol,
                                XmNshadowType, XmSHADOW_OUT,
                                NULL);

    leftSide = XtVaCreateWidget ("leftSide", xmRowColumnWidgetClass,
                                frame,
                                XmNorientation, XmVERTICAL,
                                NULL);

    str_list = (XmStringTable)XtMalloc (n * sizeof (XmString *));

    for (i=0; i<n; i++) {
        same = 0;
        for (j=0; j<i; j++)
            if (strcmp (functionName[i],functionName[j])==0) same++;
        newName[i] = MALLOC (30);
        strcpy (newName[i], functionName[i]);
        if (same>0) {
            for (j=0; j<same; j++) strcat (newName[i], " ");
            strcat (newName[i],"\0");
        }
        str_list[i] = XmStringCreateSimple (newName[i]);
    }

    for (i=0; i<n; i++) {
        functionName[i] = MALLOC (30);
        strcpy (functionName[i], newName[i]);
    }

    frame = XtVaCreateManagedWidget ("frame9",
                                xmFrameWidgetClass, leftSide,
                                XmNshadowType, XmSHADOW_OUT,
                                NULL);

    listWidgets = XtVaCreateWidget ("listWidgets",
                                xmRowColumnWidgetClass, frame,
                                NULL);


    if (fileType==PASCAL) funcText = "Procedure Displayed:";
    else funcText = "Function Displayed:";

    XtVaCreateManagedWidget (funcText,
                            xmLabelGadgetClass, listWidgets,
                            XmNfontList, font14pt,
                                NULL);

    if (n<20) numDisplayed=n;
    else numDisplayed=20;
    XtSetArg (args[0], XmNvisibleItemCount, numDisplayed);
    XtSetArg (args[1], XmNitems, str_list);
    XtSetArg (args[2], XmNitemCount, n);
    XtSetArg (args[3], XmNshadowType, XmSHADOW_OUT);

    list_w = XmCreateScrolledList (listWidgets, "funcList", args, 4);

    XtManageChild (list_w);

    if (fileType==PASCAL)
        prevButtonText = XmStringCreateSimple ("Previous Procedure");
    else prevButtonText = XmStringCreateSimple ("Previous Function");

    prevButton = XtVaCreateManagedWidget ("prevButton",
                                        xmPushButtonWidgetClass, listWidgets,
                                        XmNlabelString, prevButtonText,
                                        XmNfontList, font12pt,
                                        NULL);
    XmStringFree (prevButtonText);
    XtAddCallback (prevButton, XmNactivateCallback, PrevCallback, NULL);

    if (fileType==PASCAL)
        nextButtonText = XmStringCreateSimple ("Next Procedure");
    else nextButtonText = XmStringCreateSimple ("Next Function");

    nextButton = XtVaCreateManagedWidget ("nextButton",
                                        xmPushButtonWidgetClass, listWidgets,
                                        XmNlabelString, nextButtonText,
                                        XmNfontList, font12pt,
                                        NULL);

    XmStringFree (nextButtonText);
    XtAddCallback (nextButton, XmNactivateCallback, NextCallback, NULL);

    XtManageChild (listWidgets);


    XtVaCreateManagedWidget ("\n\n",
                            xmLabelGadgetClass, leftSide,
                            NULL);

    saveButtonText = XmStringCreateSimple ("   Save to File");
    saveButton = XtVaCreateManagedWidget ("saveButton",
                                        xmPushButtonWidgetClass, leftSide,
                                        XmNlabelString, saveButtonText,
                                        XmNfontList, font14pt,
                                        NULL);

    XmStringFree (saveButtonText);
    XtAddCallback (saveButton, XmNactivateCallback, SaveCallback, NULL);

    XtVaCreateManagedWidget ("\n\n",
                            xmLabelGadgetClass, leftSide,
                            NULL);

    quitButtonText = XmStringCreateSimple ("       Quit");

    quitButton = XtVaCreateManagedWidget ("quitButton",
                                        xmPushButtonWidgetClass, leftSide,
                                        XmNlabelString, quitButtonText,
```

```
                                XmNfontList, font14pt,
                                NULL);

    XmStringFree (quitButtonText);

    XtAddCallback (quitButton, XmNactivateCallback, QuitCallback, NULL);

    rightSide = XtVaCreateWidget ("rightSide", xmRowColumnWidgetClass,
                                  bottomRowCol,
                                  XmNorientation, XmVERTICAL,
                                  NULL);

    frame = XtVaCreateManagedWidget ("frame7",
                                     xmFrameWidgetClass, rightSide,
                                     XmNshadowType, XmSHADOW_OUT,
                                     NULL);

    commentInfo_w = XtVaCreateWidget ("commentInfo", xmRowColumnWidgetClass,
                                  frame,
                                  XmNorientation, XmVERTICAL,
                                  NULL);

    XtAddCallback (list_w, XmNbrowseSelectionCallback, selection, NULL);

    functionRowCol = XtVaCreateWidget ("functionRowCol",
                                       xmRowColumnWidgetClass,
                                       commentInfo_w,
                                       XmNorientation, XmHORIZONTAL,
                                       NULL);

    if (fileType==PASCAL) funcText = "Procedure:";
    else funcText = "Function: ";

    XtVaCreateManagedWidget (funcText, xmLabelGadgetClass,
                             functionRowCol,
                             XmNfontList, font14pt,
                             NULL);

   funcName_w = XtVaCreateManagedWidget ("funcName_w", xmTextWidgetClass,
functionRowCol,
                                         XmNcolumns, 25,
                                         XmNeditable, FALSE,
                                         XmNfontList, font14pt,
                                         NULL);


    calledByRowCol = XtVaCreateWidget ("calledByRowCol",
                                       xmRowColumnWidgetClass,
                                       commentInfo_w,
                                       XmNorientation, XmHORIZONTAL,
                                       NULL);

    XtVaCreateManagedWidget ("Called By:", xmLabelGadgetClass,
                             calledByRowCol,
                             XmNfontList, font14pt, NULL);

    calledBy_w = XtVaCreateManagedWidget ("calledBy_w", xmTextWidgetClass,
                                          calledByRowCol,
                                          XmNcolumns, 50,
                                          XmNeditable, FALSE,
                                          XmNfontList, font12pt,
                                          NULL);

    callsRowCol = XtVaCreateWidget ("callsRowCol", xmRowColumnWidgetClass,
                                    commentInfo_w,
                                    XmNorientation, XmHORIZONTAL,
                                    NULL);

XtVaCreateManagedWidget ("Calls:    ", xmLabelGadgetClass,
                         callsRowCol,
                         XmNfontList, font14pt,
                         NULL);

calls_w = XtVaCreateManagedWidget ("calls_w",
                                   xmTextWidgetClass, callsRowCol,
                                   XmNcolumns, 50,
                                   XmNfontList, font12pt,
                                   XmNeditable, FALSE,
                                   NULL);

expRowCol = XtVaCreateWidget ("expRowCol", xmRowColumnWidgetClass,
                              commentInfo_w,
                              XmNorientation, XmVERTICAL,
                              XmNalignment, XmALIGNMENT_CENTER,
                              NULL);

if (fileType==PASCAL) funcText = "Procedure Explanation:";
else funcText = "Function Explanation:";

XtVaCreateManagedWidget (funcText, xmLabelGadgetClass,
                         expRowCol,
                         XmNfontList, font14pt,
                         NULL);

XtSetArg (args[0], XmNrows, 5);
XtSetArg (args[1], XmNcolumns, 80);
XtSetArg (args[2], XmNfontList, font12pt);
XtSetArg (args[3], XmNscrollHorizontal, FALSE);
XtSetArg (args[4], XmNeditMode, XmMULTI_LINE_EDIT);
XtSetArg (args[5], XmNwordWrap, TRUE);

exp_text_w = XmCreateScrolledText (expRowCol, "exp_text_w", args, 6);

XtManageChild (functionRowCol);
XtManageChild (callsRowCol);
XtManageChild (calledByRowCol);
XtManageChild (exp_text_w);
XtManageChild (expRowCol);
XtManageChild (commentInfo_w);

lowerRowCol = XtVaCreateWidget ("lowerRowCol", xmRowColumnWidgetClass,
                                rightSide,
                                XmNorientation, XmVERTICAL,
                                NULL);


XtSetArg (textArgs[0], XmNrows, 23);
XtSetArg (textArgs[1], XmNcolumns, 80);
XtSetArg (textArgs[2], XmNfontList, font12pt);
XtSetArg (textArgs[3], XmNscrollHorizontal, TRUE);
XtSetArg (textArgs[4], XmNeditMode, XmMULTI_LINE_EDIT);
XtSetArg (textArgs[5], XmNwordWrap, TRUE);

text_w = XmCreateScrolledText (lowerRowCol, "text_w", textArgs, 6);
XtAddCallback (text_w, XmNmodifyVerifyCallback, FileModified, NULL);
startingTextPosition = 0;
XtManageChild (text_w);
XtManageChild (lowerRowCol);
```

```
    XtManageChild (rightSide);

    for (i=0; i<n; i++) XmStringFree (str_list[i]);
    XtFree (str_list);

    XtManageChild (leftSide);
    XtManageChild (bottomRowCol);
    XtManageChild (rowcol);

    if (commandLine) XtRealizeWidget (toplevel);
    else XtPopup (toplevel, XtGrabNone);

    startingTextPosition = startFuncPos[0];
    functionText = &programText[startingTextPosition];
    XmTextSetString (funcName_w, functionName[0]);
    XmTextSetString (calledBy_w, calledBy[0]);
    XmTextSetString (calls_w, calls[0]);
    XmTextSetString (exp_text_w, explanationText[0]);
    XmTextSetString (text_w, programText);
    XmTextSetInsertionPosition (text_w, 0);

    curItem = 1;
    defaultItem = XmStringCreateSimple (functionName[0]);
    XmListSelectItem (list_w, defaultItem, TRUE);

    if (commandLine) XtAppMainLoop (app);
}



/*****************************************************************************/
/* Function:  GetNewItem                                                     */
/*                                                                           */
/* Called By: selection                                                      */
/* Calls:     (none)                                                         */
/*                                                                           */
/* This function is passed the current item chosen and the function name of  */
/* that item.  It loads the widgets with the new item's information.         */
/*****************************************************************************/


void GetNewItem (curItem,choice)
int curItem;
char *choice;
{
    startingTextPosition = startFuncPos[curItem-1];

    XmTextSetString (funcName_w, choice);
    XmTextSetString (calledBy_w, calledBy[curItem-1]);
    XmTextSetString (calls_w, calls[curItem-1]);
    XmTextSetString (exp_text_w, explanationText[curItem-1]);

    XmTextSetInsertionPosition (text_w, startingTextPosition);
    XmTextSetTopCharacter (text_w, startingTextPosition);
}



/*****************************************************************************/
/* Function:  CheckVisibility                                                */
/*                                                                           */
/* Called By: NextCallback, PrevCallback                                     */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Checks to see if the newly selected item is visible in the list; if it    */
/* isn't, the list is scrolled so that the item will appear.                 */
/*****************************************************************************/

void
CheckVisibility (curItem)
int curItem;
{
    int top,visible;
    XmString newItem;

    newItem = XmStringCreateSimple (functionName[curItem-1]);

    XtVaGetValues (list_w,
                   XmNtopItemPosition, &top,
                   XmNvisibleItemCount, &visible,
                   NULL);

    if (curItem < top) XmListSetPos (list_w, curItem);
    else if (curItem >= top+visible)
        XmListSetBottomPos (list_w, curItem);
    XmListSelectItem (list_w, newItem, TRUE);
}



/*****************************************************************************/
/* Function:  NextCallback                                                   */
/*                                                                           */
/* Called By: StartUI                                                        */
/* Calls:     CheckVisibility                                                */
/*                                                                           */
/* Callback for the "Next Function" pushbutton.                              */
/*****************************************************************************/

void
NextCallback (w, client_data, cbs)
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *cbs;
{
    char *choice;

    if (curItem<totalFunctions) CheckVisibility (curItem+1);
}



/*****************************************************************************/
/* Function:  PrevCallback                                                   */
/*                                                                           */
/* Called By: StartUI                                                        */
/* Calls:     CheckVisibility                                                */
/*                                                                           */
/* Callback for the "Previous Function" pushbutton.                          */
/*****************************************************************************/
```

```
void
PrevCallback (w, client_data, cbs)
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *cbs;
{
    XmString newItem;
    char *choice;

    if (curItem>1) CheckVisibility (curItem-1);
}
```

```
/*****************************************************************************/
/* Function:  selection                                                      */
/*                                                                           */
/* Called By: StartUI                                                        */
/* Calls:     GetNewItem                                                     */
/*                                                                           */
/* Callback for the list widget.  This is the function that is called when   */
/* the user selects a new function from the list.  It is also called when    */
/* the user pushes either the "Previous Function" or the "Next Function"     */
/* pushbutton.                                                               */
/*****************************************************************************/

void selection (list_w, client_data, cbs)
Widget list_w;
XtPointer client_data;
XmListCallbackStruct *cbs;
{
    char *choice;

    XmStringGetLtoR (cbs->item, charset, &choice);

    if (curItem!=0) explanationText[curItem-1] = XmTextGetString (exp_text_w);

    curItem = cbs->item_position;
    GetNewItem (curItem,choice);
    XtFree (choice);
}
```

```
/*****************************************************************************/
/* Function:  QuitCallback                                                   */
/*                                                                           */
/* Called By: StartUI                                                        */
/* Calls:     Exit                                                           */
/*                                                                           */
/* This is the callback function for the "Quit" pushbutton on the main       */
/* window.                                                                   */
/* Destroys the dialog, but only after a popup asks the user if he really    */
/* wants to do it.                                                           */
/*                                                                           */
/*****************************************************************************/
```

```
void
QuitCallback (w, client_data, cbs)
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *cbs;
{
    static Widget dialog;
    extern void Exit();
    XmString label;

    dialog = XmCreateMessageDialog (w, "message", NULL, 0);

    label = XmStringCreateLtoR ("Quit -- Are You Sure?",
                                XmSTRING_DEFAULT_CHARSET);

    XtVaSetValues (dialog, XmNdialogType, XmDIALOG_MESSAGE,
                XmNmessageString, label, NULL);

    XtManageChild (dialog);
    XtAddCallback (dialog, XmNokCallback, Exit, w);
    XtAddCallback (dialog, XmNcancelCallback, XtDestroyWidget, NULL);
}
```

```
/*****************************************************************************/
/* Function:  DoCommentBox                                                   */
/*                                                                           */
/* Called By: Save                                                           */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Calls routines in xcommentbase.c which stick the desired information into */
/* the comment area.                                                         */
/*****************************************************************************/

void DoCommentBox (j,outfile)
int j;
FILE *outfile;
{
    int index, nameLen;

    if (fileType!=CPP) fputs (beginComment,outfile);

    if ((boxComments) && (fileType!=CPP)) {
        for (index=5; index<colwidth; index++) fputs("*",outfile);
        fputs (endComment,outfile);
    }
    if (fileType!=CPP) fputs ("\n",outfile);

    if (j>0) {
        if (1 & commentInfo) writeheader (functionName[j-1],j,outfile);
        addgeneratedlines (j,outfile);
    }

    /* stick bottom of comment box in file... */

    if ((boxComments) && (fileType!=CPP)) {
        fputs (beginComment,outfile);
        for (index=5; index<colwidth; index++) fputs("*",outfile);
    }
```

```
    else if (fileType==CPP) fputs (beginComment,outfile);
    else if (fileType==C) fputs (" ",outfile);

    fputs (endComment,outfile);
    fputs ("\n\n",outfile);
}



/*****************************************************************************/
/* Function:  SaveCallback                                                   */
/*                                                                           */
/* Called By: StartUI                                                        */
/* Calls:     CancelCallback, CheckBoxCallback, Save, Toggled                */
/*                                                                           */
/* Callback for the "Save To File" pushbutton.  Creates a popup widget which */
/* is positioned directly under the Save To File pushbutton.  The user can   */
/* toggle between box and "regular" comments, and he can also choose exactly */
/* which information he wants to be saved to the file.  In addition, if he   */
/* wants the output to be a different file than the default file (the        */
/* original filename), he may change that in the given input box.            */
/*****************************************************************************/

void
SaveCallback (w)
Widget w;
{
    Arg args[3];
    Widget radioBox, frame, saveInfo, toggleBox, saveButton2, lowestSave,
           leftSave, rightSave, cancelButton, lowerSave, title, label_w,
            funcName_t, calledBy_t, calls_t, exp_t, lowestRowcol;
    XmString saveButtonText, cancelButtonText, str, underDec, aboveDec;
    void Save();
    void Toggled();
    void CheckBoxCallback();
    void CancelCallback();
    Position xLoc, yLoc, topX, topY;
    Dimension width, height, dHeight;
    char *funcText;

    XtRealizeWidget (toplevel);

    if (!dialogUp) {

        dialogUp = TRUE;

        XtVaGetValues (saveButton, XmNx, &xLoc, XmNy, &yLoc, NULL);
        XtVaGetValues (toplevel, XmNx, &topX, XmNy, &topY, NULL);

        dialog = XtVaCreatePopupShell ("popup",
                                       xmDialogShellWidgetClass, XtParent(w),
                                       XmNtitle, "Save To File",
                                       XmNdeleteResponse, XmDESTROY,
                                       NULL);

        XtVaSetValues (dialog, XmNx, xLoc+topX, XmNy, yLoc+topY+75, NULL);

        saveInfo = XtVaCreateWidget ("saveInfo",
                                     xmRowColumnWidgetClass, dialog,
                                     XmNorientation, XmVERTICAL,
                                     NULL);

        title = XtVaCreateWidget ("title", xmPanedWindowWidgetClass,
                                  saveInfo,
                                  XmNorientation, XmHORIZONTAL,
                                  NULL);

        frame = XtVaCreateManagedWidget ("frame10",
                                         xmFrameWidgetClass, title,
                                         XmNshadowType, XmSHADOW_OUT,
                                         NULL);

        str = XmStringCreateSimple ("Save Options");

        label_w = XtVaCreateManagedWidget ("label_w",
                                           xmLabelWidgetClass, frame,
                                           XmNlabelString, str,
                                           XmNfontList, font14pt,
                                           NULL);
        XtManageChild (title);

        lowerSave = XtVaCreateWidget ("lowerSave",
                                      xmRowColumnWidgetClass, saveInfo,
                                      XmNorientation, XmHORIZONTAL,
                                      NULL);

        leftSave = XtVaCreateWidget ("leftSave",
                                     xmRowColumnWidgetClass, lowerSave,
                                     XmNorientation, XmVERTICAL,
                                     NULL);

        XtVaCreateManagedWidget ("Comment Type:",
                                 xmLabelGadgetClass, leftSave,
                                 XmNfontList, font14pt,
                                 NULL);

        frame = XtVaCreateManagedWidget ("frame1",
                                         xmFrameWidgetClass, leftSave,
                                         XmNshadowType, XmSHADOW_OUT,
                                         NULL);

        aboveDec = XmStringCreateSimple ("Regular Comments");
        underDec = XmStringCreateSimple ("Box Comments");

        radioBox = XmVaCreateSimpleRadioBox(frame, "radioBox",
                                 boxComments, Toggled,
                                 XmVaRADIOBUTTON, aboveDec, NULL, NULL, NULL,
                                 XmVaRADIOBUTTON, underDec, NULL, NULL, NULL,
                                 NULL);

        XmStringFree (aboveDec);
        XmStringFree (underDec);

        XtManageChild (radioBox);

        rightSave = XtVaCreateWidget ("rightSave",
                                      xmRowColumnWidgetClass, lowerSave,
                                      XmNorientation, XmVERTICAL,
                                      NULL);

        XtVaCreateManagedWidget ("Include Information:",
                                 xmLabelGadgetClass, rightSave,
                                 XmNfontList, font14pt,
                                 NULL);

        frame = XtVaCreateManagedWidget ("frame2",
```

```
                                    xmFrameWidgetClass, rightSave,
                                    XmNshadowType, XmSHADOW_OUT,
                                    NULL);

    toggleBox = XtVaCreateWidget ("toggleBox",
                                  xmRowColumnWidgetClass, frame,
                                  NULL);

    if (fileType==PASCAL) funcText = "Procedure Name";
    else funcText = "Function Name";

    funcName_t = XtVaCreateManagedWidget (funcText,
                                          xmToggleButtonWidgetClass,
                                          toggleBox,
                                          NULL);

    XtAddCallback (funcName_t, XmNvalueChangedCallback,
                   CheckBoxCallback, 0);

    calledBy_t = XtVaCreateManagedWidget ("Called By Line",
                                          xmToggleButtonWidgetClass,
                                          toggleBox,
                                          XmNradioBehavior, TRUE,
                                          NULL);

    XtAddCallback (calledBy_t, XmNvalueChangedCallback,
                   CheckBoxCallback, 1);

    calls_t = XtVaCreateManagedWidget ("Calls Line",
                                       xmToggleButtonWidgetClass,
                                       toggleBox,
                                       NULL);

    XtAddCallback (calls_t, XmNvalueChangedCallback, CheckBoxCallback, 2);

    if (fileType==PASCAL) funcText = "Procedure Explanation";
    else funcText = "Function Explanation";

    exp_t = XtVaCreateManagedWidget (funcText,
                                     xmToggleButtonWidgetClass,
                                     toggleBox,
                                     NULL);

    XtAddCallback (exp_t, XmNvalueChangedCallback, CheckBoxCallback, 3);

    XtManageChild (toggleBox);

    if (1 & commentInfo) XmToggleButtonSetState (funcName_t, TRUE, FALSE);
    if (2 & commentInfo) XmToggleButtonSetState (calledBy_t, TRUE, FALSE);
    if (4 & commentInfo) XmToggleButtonSetState (calls_t, TRUE, FALSE);
    if (8 & commentInfo) XmToggleButtonSetState (exp_t, TRUE, FALSE);

    saveButtonText = XmStringCreateSimple ("    Do It");
    saveButton2 = XtVaCreateManagedWidget ("saveButton",
                                       xmPushButtonWidgetClass, leftSave,
                                       XmNlabelString, saveButtonText,
                                       XmNfontList, font14pt,
                                       NULL);

    XmStringFree (saveButtonText);
    XtAddCallback (saveButton2, XmNactivateCallback, Save, NULL);

    cancelButtonText = XmStringCreateSimple ("   Cancel");

    cancelButton = XtVaCreateManagedWidget ("cancelButton",
                                       xmPushButtonWidgetClass, leftSave,
                                       XmNlabelString, cancelButtonText,
                                       XmNfontList, font14pt,
                                       NULL);

    XtAddCallback (cancelButton, XmNactivateCallback,
                   CancelCallback, NULL);

    lowestSave = XtVaCreateWidget ("lowestSave",
                                   xmPanedWindowWidgetClass, saveInfo,
                                   XmNorientation, XmHORIZONTAL,
                                   NULL);

    frame = XtVaCreateManagedWidget ("frame16",
                                     xmFrameWidgetClass, lowestSave,
                                     XmNshadowType, XmSHADOW_OUT,
                                     NULL);

    lowestRowcol = XtVaCreateWidget ("lowestRowcol",
                                     xmRowColumnWidgetClass, frame,
                                     XmNorientation, XmHORIZONTAL,
                                     XmNalignment, XmALIGNMENT_CENTER,
                                     NULL);

    XtVaCreateManagedWidget (" Filename:",
                             xmLabelGadgetClass, lowestRowcol,
                             XmNfontList, font14pt,
                             XmNcolumns, 30,
                             NULL);

    filenameText = XtVaCreateManagedWidget ("filenameText",
                                            xmTextWidgetClass,
                                            lowestRowcol, NULL);

    XmStringFree (cancelButtonText);
    XtManageChild (leftSave);
    XtManageChild (rightSave);
    XtManageChild (lowerSave);
    XtManageChild (lowestRowcol);
    XtManageChild (lowestSave);
    XtManageChild (saveInfo);

    XtManageChild (dialog);
    XmTextSetString (filenameText, shortFn);

    XtPopup (dialog, XtGrabNone);
  }
}


/*****************************************************************************/
/* Function:  Toggled                                                        */
/*                                                                           */
/* Called By: SaveCallback                                                   */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Callback for the toggle widget; sets the variable boxComments to          */
/* whichever type the user has selected.                                     */
/*****************************************************************************/
```

```
void Toggled (widget, which, state)
Widget widget;
int which;
XmToggleButtonCallbackStruct *state;
{
    if (state->set) {
        boxComments = which;
    }
}



/*****************************************************************************/
/* Function:  CheckBoxCallback                                               */
/*                                                                           */
/* Called By: SaveCallback                                                   */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Callback for the CheckBox widget.  Sets the variable commentInfo to the   */
/* bitpattern which represents the information that the user desires to      */
/* write to the file.                                                        */
/*****************************************************************************/


void CheckBoxCallback (widget, bit, state)
Widget widget;
int bit;
XmToggleButtonCallbackStruct *state;
{
    if (state->set) commentInfo |= (1<<bit);
    else commentInfo &= ~(1<<bit);
}




/*****************************************************************************/
/* Function:  Save                                                           */
/*                                                                           */
/* Called By: SaveCallback                                                   */
/* Calls:     DoCommentBox                                                   */
/*                                                                           */
/* Callback for the "Do It" pushbutton widget in the Save popup widget.      */
/* Writes everything to the file by calling DoCommentBox in between segments */
/* of the program text.                                                      */
/*****************************************************************************/

void
Save (w)
Widget w;
{
    FILE *outfile, *infile, *tempFile;
    int i, j, curLine, num, curPos, lineLen;
    char line[LINELEN];
    char *saveFn;
    Boolean done, readNextLine;

    explanationText[curItem-1] = MALLOC (XmTextGetLastPosition (exp_text_w));
    if (curItem!=0) explanationText[curItem-1] = XmTextGetString (exp_text_w);

    tempFile = fopen("xcomment.tmp2","w");
    programText = XmTextGetString (text_w);
    fputs (programText,tempFile);
    fclose (tempFile);
    infile = fopen ("xcomment.tmp2","r");
    saveFn = XmTextGetString (filenameText);
    if (strlen (saveFn)==0) saveFn = oldFn;
    outfile = fopen(saveFn,"w");
    curLine = 0;
    if (fileType==CPP) num=4;
    else num=2;
    curPos = 0;
    done = FALSE;
    for (j=1; j<totalFunctions; j++) {
        if (15 & commentInfo) DoCommentBox (j,outfile);
        if (done) readNextLine = FALSE;
        else readNextLine = TRUE;
        done = FALSE;
        while ((curPos < startFuncPos[j]) && (!done)) {
            if (readNextLine) {
                fgets (line,LINELEN,infile);
                lineLen = strlen (line);
            }
            if (curPos+lineLen < startFuncPos[j]) {
                fputs (line,outfile);
                curPos += lineLen;
                curLine++;
                readNextLine = TRUE;
            }
            else done=TRUE;
        }
    }

    /* get the last function... */

    if (15 & commentInfo) DoCommentBox (j,outfile);
    if (!done) fgets (line,LINELEN,infile);
    while (!feof(infile)) {
        fputs (line,outfile);
        fgets (line,LINELEN,infile);
    }
    fclose (outfile);
    fclose (infile);
    unlink ("xcomment.tmp2");
    if (fileType==CPP) unlink ("xcomment.c");

    XtDestroyWidget (dialog);
    dialogUp = FALSE;

}



/*****************************************************************************/
/* Function:  CancelCallback                                                 */
/*                                                                           */
/* Called By: SaveCallback                                                   */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Callback for the "Cancel" pushbutton widget in the Save popup widget.     */
/* Destroys the Save popup widget.                                           */
/*                                                                           */
/*****************************************************************************/
```

```
void
CancelCallback (w)
Widget w;
{
    XtDestroyWidget (dialog);
    dialogUp = FALSE;
}




/*****************************************************************************/
/* Function:  Exit                                                           */
/*                                                                           */
/* Called By: QuitCallback, StartUI                                          */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Called by the "Quit" pushbutton widget in the main window.  Destroys the  */
/* main window dialog, and if the user started from the command line,        */
/* totally exits the program.  Otherwise, the control will be transerred to  */
/* the file widget in the file "xcommentFile.c".                             */
/*****************************************************************************/


void Exit (w, client_data, cbs)
Widget w;
XtPointer client_data;
XmPushButtonCallbackStruct *cbs;
{
    if (fileType==CPP) rename("xcomment.c",oldFn);
    XtDestroyWidget (toplevel);
    dialogUp = FALSE;
    if (commandLine) exit (0);
}




/*****************************************************************************/
/* Function:  FileModified                                                   */
/*                                                                           */
/* Called By: StartUI                                                        */
/* Calls:     (none)                                                         */
/*                                                                           */
/* Callback routine for whenever the program text is modified.  Calculates   */
/* how much text was either inserted or deleted, in which function the       */
/* modification took place, and adjusts the starting lines and positions of  */
/* each function which follows in the file.                                  */
/*****************************************************************************/


void FileModified (text_ww, unused, cbs)
Widget text_ww;
XtPointer unused;
XmTextVerifyCallbackStruct *cbs;
{
    char *text, *deletedText, *startDeletion;
    int i,j,charsDeleted,linesDeleted;

    /* check to see where we are in file...*/
    i=0;
    while (cbs->startPos > startFuncPos[i]) i++;

        if (cbs->text->ptr==NULL) {
            charsDeleted = cbs->endPos - cbs->startPos;
            text = XmTextGetString (text_ww);
            deletedText = MALLOC (charsDeleted);
            startDeletion = &text[cbs->startPos];
            strncpy (deletedText, startDeletion, charsDeleted);
            linesDeleted = 0;
            for (j=0; j<charsDeleted; j++)
                if (deletedText[j]==RETURN) linesDeleted++;
        }

        for (j=i; j<totalFunctions; j++) {
            if (cbs->text->ptr!=NULL) {
                startFuncPos[j]++;
                if (cbs->text->ptr[0]==RETURN) startLine[j]++;
            }
            else {
                startFuncPos[j] -= charsDeleted;
                startLine[j] -= linesDeleted;
            }
        }
}
```