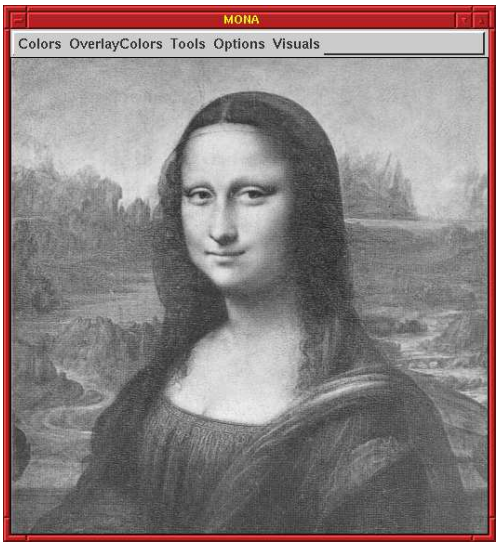
Image Processing Laboratory
Department of Informatics
University of Oslo

Report No. 92



XITE

X-based Image Processing Tools and Environment

Programmer's Manual

For version 3.4

Svein Bøe

June 1998

Bildebehandlingslaboratoriet
Institutt for informatikk
Universitetet i Oslo

Image Processing Laboratory
Department of Informatics
University of Oslo

Tittel/Title:

XITE Programmer's Manual

Forfatter(e)/Author(s):

Svein Bøe

Rapport nr./Report no.: **92** ISBN: **82-7476-061-1** Dato/Date: **June 1998**

Resymé/Abstract:

XITE consists of display programs with image widget and graphical user interface as well as more than 200 command line programs and 600 subroutines for image processing, all documented on-line.

The command line programs and subroutine library are written in C and run under UNIX and Windows.

The display programs run under UNIX. They work with images of arbitrary size and pixel type on 8-bit PseudoColor and 24-bit DirectColor and TrueColor X11 displays. Images can be zoomed and panned, and colortables can be selected from a menu. The main display program, xshow, gives access to most of the other command line programs via a menu interface which the user can customize and extend to include local programs. Input images for the menu entries can be selected with the mouse, and output images appear on the display.

This report describes how to write applications using the XITE routine library and ximage toolkit.

Norske emneord/Indexing terms - Norwegian:

Bildebehandling
Bildebegrep
Bildeprogram
Vindussystemet X
Farger
UNIX
C
Windows

Engelske emneord/Indexing terms - English:

Image Processing
Image Concept
Display Program
X Window System
Colors
UNIX
C
Windows

Adresse:

Bildebehandlingslaboratoriet
Institutt for informatikk
Universitetet i Oslo
Boks 1080 Blindern
0316 Oslo

epost: blab@ifi.uio.no
tlf: 22 85 24 10

Address:

Image Processing Laboratory
Department of Informatics
University of Oslo
P. O. Box 1080 Blindern
N – 0316 Oslo
NORWAY

email: blab@ifi.uio.no
phone: +47 22 85 24 10

XITE

X-based Image processing Tools and Environment

Programmer's Manual

For version 3.4

Image Processing Laboratory
Department of Informatics
University of Oslo

Svein Bøe
June 1998

Contents

1 Introduction

Unfortunately, this document is still in a preliminary state.

XITE (pronounced *excite*) is an acronym for "X-based Image processing Tools and Environment". XITE is developed by the Image Processing Laboratory, Department of Informatics, University of Oslo, Norway.

This document contains information on how to write C programs which use the XITE routine library. There are also guidelines on how to adhere to the programming standard for XITE in case you wish to contribute routines and programs to XITE itself, and not merely use what is already available.

For information on how to use XITE without C programming, please refer to the User's Manual [4].

In this text, the symbol `$XITE_HOME` represents the XITE home directory. This could be e.g. `/usr/local/xite`, `~xite` or whatever your local XITE administrator or system manager chose when XITE was installed on your system. For installation of XITE, refer to the [3, System Administrator's Manual].

More information on all the programs and functions mentioned in this document, can be found in the online hypertext Reference Manual [2].

2 Compilation and linking of XITE dependent programs

The script cxite should be used to compile and link C source files which use functions and definitions from XITE. This script knows where XITE header files and libraries are located, as well as X Window System header files and libraries. Try the command

```
§ cxite -man
```

for help on cxite.

3 Reading and writing BIFF images

The XITE library contains several functions for reading and writing BIFF (Blab Image File Format) images. The main BIFF routines are listed in table 1 on the following page. Always use the reading and writing routines from the table when reading from file or writing images to file. These routines will interpret correctly the special file names "`-`" (for standard input or standard output), "`-0`" (for standard input) and "`-1`" (for standard output). Refer to subsection 4.4 for details on this. The functions also swap bytes when an image is read on a platform with a different byte ordering than the byte ordering for the platform which wrote the image to file.

3.1 Tracking the image history

There is a function Ihistory which can be used to track the processing history of an image. A call to this function will append a line to the text field of an image. This text or history field can be printed e.g. with the program biffinfo or inside the XITE display programs.

If an image is the result of processing another image, the function Icopy_text can be used to copy the history/text field from the old image to the new image. A subsequent call to Ihistory will

Function name	Effect
Iread_image	Open BIFF image file and read image.
Iwrite_image	Open BIFF image file and write image.
Iread_band	Read image band from open file.
Iwrite_band	Write image band to open file.
Iopen_image	Open BIFF image file for reading and/or writing.
Iclose_image	Close BIFF image file.
Imake_image	Create complete image data structure, including bands and pixel value storage. Pixel values are not initialized.
Init_image	Only initialize image info and pointers. Don't create bands.
Imake_band	Create complete band data structure, including pixel value storage. Pixel values are not initialized.
Imake_bands	Allocate memory for all image bands. Pixel values are not initialized.
Init_band	Only initialize band info. Don't allocate storage for pixel values.
Init_bands	Initialize all bands of image. Don't allocate storage for pixel values.

Table 1: Main BIFF routines for image and band handling.

append to the image history.

4 Standardized command line user interface

As described in the User's Manual [4], all XITE programs are supposed to behave consistently when processing input arguments. The various conventions are described below.

4.1 On-line help with program options

When a program is fed one of the options `-help`, `-usage`, `-man`, `-whatis` or `-verbose`, the behavior for each option should be

`-help`, `-usage`: Give a usage message for the program and terminate.

`-man`: Give the same as `man program`. The manual page is extracted from a comment written in the program source code. This is described in section 6.

`-whatis`: Give a one line description of the program, and terminate. This one line description is collected from the same comment as the one mentioned for option `-man` above.

`-verbose`: Make some programs report the actions taking place.

The above is accomplished by using the function InitMessage. This function will save the program name and a usage text and search the command line for one of the options. The call to InitMessage should always be one of the first statements in a program. The emacs command `blab-header-P` will insert a template for a call to InitMessage (refer to subsection 6.1). It looks something like this

```
InitMessage(&argc, argv, xite_app_std_usage_text(
  "Usage: %s <required option>... [<optional option>...]\n\
          <inimage> <outimage>\n"));
```

The programmer should fill in the options for the particular program. The call in InitMessage to xite_app_std_usage_text appends a description of the standard XITE help options mentioned above to the usage message.

4.2 Message feedback from your programs to the user

The function InitMessage is one of a collection of XITE message handling functions. They provide a standard way of informing the user on different levels.

The function Usage should be called if the user gives illegal options or arguments, or if necessary options or filename arguments are missing. It will print a supplied error message, followed by the usage text which was installed by InitMessage, and finally terminate the program. The functions Info, Message, Warning and Error will possibly print a message and possibly terminate the program, depending on the values of various environment variables, program options and function arguments given.

4.3 Message feedback from the BIFF library functions to the user

The functions which take care of image input/output and other tasks related to the BIFF file format, have their own simple message system. If you want errors encountered in these file format functions to be printed, call the function Iset_message in the main program. If you also want the program to terminate if such an error is encountered, then call the function Iset_abort in the main program. The emacs command `blab-header-P` will insert templates for calls to Iset_message and Iset_abort (refer to subsection 6.1).

4.4 Piping and input/output images

All XITE programs should be designed in such a way that they expect file names to be given as arguments. They should not wait for input to appear on the standard input. Rather, they should print a usage message (with the function Usage) if a filename argument is missing. They may be told explicitly to take input from standard input by giving the filename "`-`" or "`-0`" on the command line. Output should be sent to standard output if the output filename is "`-`" or "`-1`". When reading or writing BIFF images or BIFF colortables, these filenames are recognized by the BIFF format I/O routines, such as Iread_image and Iwrite_image (refer to section 3). Therefore, always read and write BIFF images with XITE library functions.

4.5 Processing program options

As mentioned above, some options are processed by InitMessage. Other options should be processed by read_switch (for text options) and its cousins read_bswitch (for true/false or on/off options), read_iswitch (for integer options) and read_dswitch (for double float options).

5 Example of a main program

A typical main program in XITE, with the ingredients from the above section 3 and section 4, will look something like this (an excerpt from `crossSection.c` in the XITE source directory `arithmetic`).

```
/*P:crossSection*

-------------------------------------------------------------

                crossSection
-------------------------------------------------------------

Name:           crossSection - Find cross sections, row or column of image

Syntax:         crossSection [<option>...] <inimage> <outimage>

Description:    'crossSection' reads a BIFF image and finds a cross-section,
                row or column.

Options:        &-M
                cross-section maximum (default)

                &-n num
                pick row or column number 'num'

                &-t title
                Title of output image

                &-s scale
                Scale curve height relative to image height (default is 1.0,
                which means that the curve peak will touch the top edge of the
                image).

See also:       profile(1), crossSection(3)

Return value:   2 => Illegal arguments.

Author:         Svein Bøe

Id:             crossSection.c,v 1.20 1995/01/17 14:38:52 svein Exp
-------------------------------------------------------------

*/

#include <xite/includes.h>
#include <xite/message.h>
#include <xite/biff.h>
#include <xite/readarg.h>

#ifdef MAIN

#ifndef FUNCPROTO
int main(argc, argv)
int argc;
char **argv;
#else /* FUNCPROTO */
int main(int argc, char **argv)
#endif /* FUNCPROTO */
{
  int maxi, num;
  char *title, *outFile, *options;
  double scale;
```

```
  Iset_message(1);
  Iset_abort(1);
  InitMessage(&argc, argv, xite_app_std_usage_text(
    "Usage: %s [<option>...] <inimage> <outimage>\n\
       where <option> is chosen from\n\
         -M          : Cross-section maximum (default) \n\
         -n <num>    : Pick row or column number <no> \n\
         -t <title>  : Title of output image\n\
         -s <scale>  : Scale curve height relative to image height\n"));

  /* Standard behaviour when no arguments or options. */
  if (argc == 1) Usage(1, NULL);

  /* Save the command line arguments (other than the standard XITE help options). */
  options = argvOptions(argc, argv);

  /* Process options. */
  maxi   = read_bswitch(&argc, argv, "-M");
  num    = read_iswitch(&argc, argv, "-n", -1);
  title  = read_switch(&argc,  argv, "-t", 1, "");
  scale  = read_dswitch(&argc, argv, "-s", 1.0);

  /* Check number of arguments. */
  if (argc != 3) Usage(2, "Illegal number of arguments.\n");

  /* Read input image argument */
  img     = Iread_image(argv[1]);
  outFile = argv[2];

  /* Create IMAGE out_img and process */

  /* Copy text field (image history) from input image to output image. */
  Icopy_text(img, out_img);

  /* Append description of current processing to image history. */
  Ihistory(out_img, argv[0], options);

  /* Write result to file */
  Iwrite_image(out_img, outFile);

  return(0);
}

#endif
```

6 Documentation and manual pages

If the main comments for functions and programs are written in a particular style, a hypertext reference manual and manual pages for the UNIX man command may be generated automatically. Examples of such comments may be found in the file

`$XITE_HOME/src/arithmetic/scale.c`.

A main program comment is also shown above in section 5.

6.1 Emacs commands for main comments

You get access to emacs commands for the various types of comments by inserting the line

```
(load-file "$XITE_HOME/etc/emacs-header")
```

into your `.emacs` file. Next time you start emacs, you may type `M-x` followed by one of the commands from the list below.

`blab-header-P:` Used for the main comment of a program.

Also refer to subsection 4.1, subsection 6.2 and subsection 6.3.

`blab-header-F:` Used for the main comment of a function. Remember to specify the header files which are necessary to get access to the declarations required for use of this function. This is specified in the "Syntax" part of the comment and will typically look something like

```
#include <xite/region.h>
```

if the declarations are contained in `region.h` under the XITE `include` directory (refer to appendix A.6). Also refer to subsection 6.3.

`blab-header-I:` Used for the main comment in a header file.

`blab-header-L:` Used for the main comment of a local function, i.e. a function which is not to be included in XITE as an independent function. No manual page will be generated in this case. A local function is most often declared static in the C source code.

`blab-header-S:` Same as `blab-header-S`, but for shell scripts (or macros).

`blab-header-H:` This does not insert a comment, but adds preprocessor directives for inclusion of some standard XITE header files.

```
#include <xite/includes.h>  /* For _XITE_PARAMS, _XITE_CPLUSPLUS_BEGIN,
                             * _XITE_CPLUSPLUS_END and constant names for
                             * header files.                              */
#include <xite/message.h>   /* For InitMessage(), Usage(), Error(),
                             * Warning() etc.                             */
#include <xite/readarg.h>   /* For argvOptions(), read_switch() etc.      */
#include <xite/biff.h>      /* For Iread_image() etc.                     */
```

`insert-header:` Used to choose one of the commands above.

On the manual page line starting with `See also:`, other related XITE programs or functions of interest may be listed. Each XITE program, function or format reference in the list should end with `(n)`, where `n` referes to a manual section number, typically 1 (for program), 3 (for routine) or 5 (for BIFF format routine). This is necessary in order to get correct hypertext links in the hypertext reference manual. An example is

```
See also:       absDiff(1), signDiff(1), multiply(1), divide(1)
```

6.2 Preprocessor directive in the main comment of a main program

The command `blab-header-P` above differs somewhat from the rest in that a directive (for the C preprocessor) is inserted in addition to a comment template. The directive contains a line with the text #ifdef MAIN and a matching line with the text #endif. The main program (the function main in C) and everything else which only main needs to know about, is supposed to be typed in between these two lines. When compiling a program (refer to section 2), the option `-DMAIN` must be given to the compiler. This is to avoid inclusion of the function main in the XITE library `libxite.a` when the other functions are compiled and archived.

6.3 Specification of the manual page name

For the emacs commands `blab-header-P`, `blab-header-F` and `blab-header-S`, the first line of the comment template will look something like

```
/*P:name*
```

(or with `F` instead of `P` and `#` prepended in the case of `blab-header-S`). The text name will be replaced by the text typed when prompted by the command. This is typically the name of the program or the function which is about to be documented.

The chosen name will become the name of the manual page for the program or function. However, name can also be specified like this

```
/*P:name1=name2*/
```

(or with `F` instead of `P`). In this case, the manual page for name1 will become a link to the manual page for name2. The actual comment for name2 must be given in the form starting with `/*P:name*`. (Remember to end the link kind of specification with the character "/", which together with "`*`" terminates this comment instruction.)

On the manual page comment line starting with `Name:`, name2 should always be listed first when the linking described above is used. This ensures that the hypertext reference manual, which is generated automatically, will also be able to follow these links.

It is also possible to write several lines as shown above in one file, but with different strings for name or name1 and name2 in each line. In this case, more manual pages will be generated, possibly as several links to the same manual page. This is useful when several programs or functions can be explained by the same manual page. Refer to the file `$XITE_HOME/src/utils/readswitch.c` for an example. Refer to the documentation for cdoc for more information.

The automatic generation of manual pages is taken care of by the installation programs, as explained in the System Administrator's Manual [3].

7 BIFF Image Concept and File Format

Some information about the BIFF Image Concept and File Format can be found in the User's Manual [4]. A detailed description is given in the BIFF Manual [5].

8 ximage toolkit for X applications

ximage is a toolkit for design of X applications in XITE, using the Image and ImageOverlay widgets. This gives the programs a common look and feel. Please refer to the Reference Manual [2] for more information about ximage.

9 Contributing to XITE

Send an email to *blab@ifi.uio.no* if you want your software to be included with the XITE distribution.

A Portability

XITE should compile at least on the following platforms

- Linux
- Windows

To simplify the use of XITE on various architectures, it is important to consider portability already during program development. This becomes even more important when both 64- and 32-bits systems are available. Some hints on reducing the number of problems are described below.

A.1 Traditional C and ANSI C

Perhaps the most important difference between traditional C (Kernighan and Ritchie) and ANSI C, is how the function heads are defined and declared. An example of a *definition* in traditional C and ANSI C respectively, is

Traditional C:

```
double f(a);
int a;
{
...
}
```

ANSI C:

```
double f(int a);
{
...
}
```

The *declaration* of the same function in traditional C and ANSI C respectively, is

Traditional C:

```
double f();
```

ANSI C:

```
double f(int a);
```

In ANSI C, the parameter type is specified in the declaration, so that the compiler can check for type consistency between formal and actual parameter. This kind of declaration is called a function prototype. (The parameter name, in this case `a`, can be omitted from the ANSI *declaration*.)

Since ANSI C improves type checking by using prototypes, one ought to write all programs using ANSI C and an ANSI C compiler. However, some users do not yet have access to an ANSI C compiler. You should therefore define and declare functions both for traditional C and ANSI C. This can be done as shown below (the function body is the same in both cases). Here is the *definition*

```
#ifdef FUNCPROTO
double f(int a)
#else /* FUNCPROTO */
double f(a)
int a;
#endif /* FUNCPROTO */
{
...
}
```

and here is the *declaration*

```
#ifdef FUNCPROTO
extern double f(int a);
#else /* FUNCPROTO */
extern double f();
#endif /* FUNCPROTO */
```

An ANSI compiler will read the prototypes, while a traditional compiler will read the traditional code. The compiler may be forced to read the prototypes by using the compiler option `-DFUNCPROTO` (although the compiler may be too old to understand this ANSI standard code). Using the option `-UFUNCPROTO` will force it to read the traditional code.

The above conditionals for the function definitions are inserted automatically by the emacs commands described in subsection 6.1.

If a program is already written in a file called `filename.c`, using traditional C, one may get the ANSI C code inserted automatically by issuing the following commands

```
% protoize -c '-I/local/xite/include' filename.c
% diff -DFUNCPROTO filename.c.save filename.c > filename.c.new
% mv filename.c.new filename.c
% rm filename.c.save
% sed 's/#else FUNCPROTO/#else \/\* FUNCPROTO \*\//' filename.c | \
  sed 's/#endif FUNCPROTO/#endif \/\* FUNCPROTO \*\//' > filename.c.sed
% mv filename.c.sed filename.c
```

If you already have the ANSI source and require traditional C, substitute the command protoize by unprotoize and give the two filename arguments of the diff command in opposite order.

If the source file contains a function main surrounded by #ifdef MAIN, #endif, you will need to rerun the above with the argument `-DMAIN` added between the quotes of the `-c` option. The same goes for other conditionally compiled segments of the source code.

If protoize is not recognized by your system, ask your system manager where you can find it (perhaps it will be in `/local/gnu/bin` or `/usr/local/gnu/bin/`).

In XITE header files, the function declarations don't look exactly like the declaration above with the FUNCPROTO macro. Instead, they use an XITE macro _XITE_PARAMS to avoid declaring the function head twice. The declaration in the header file becomes

```
extern double f _XITE_PARAMS(( int a ));
```

It is important to use two sets of parentheses around the parameter list as shown above. The macro _XITE_PARAMS is defined in the header file `xite_funcproto.h` which is included by `includes.h` and `biff.h`.

A.2 char versus unsigned char

On some platforms, a char variable is really unsigned char, on other platforms it is signed char. This means that on some platforms it may represent an integer in the range -127 to $+128$, on others the range is 0 to 255.

This platform dependency may create problems for the following source code

```
/* The preprocessor directive inserts the declaration of getchar
 * and the definition of EOF (among other things).
 * getchar is a function returning an int.
*/
#include <stdio.h>
char c;
while ( (c = getchar()) != EOF ) ;
```

The function getchar returns a character from `stdin`. It is of type int to be able to return an "illegal" character when there are no characters left, or in case of a failure. The constant EOF represents this "illegal" character, usually the integer "-1". On a platform where char is the same as unsigned char, the variable c will get the value 255 when getchar returns EOF. The loop will never stop. On a platform where char is the same as signed char, the source code will work as expected.

To avoid this problem, the variable c should be defined as an int.

A.3 Types of functions

Before you use a function in a program, the type of the function should be known, otherwise an error may occur during execution of the program. The error is due to the fact that the compiler has been given incorrect information concerning the memory space (number of bytes) to set aside for the returning function value.

The type of a function may be specified in one of several ways (remember that the term "definition" is used for the source code which specifies the whole function, including head and body, while "declaration" means that only the head is specified, i.e. the type of the function and any parameters (the latter only for ANSI C)). In the examples below, the source code is written in traditional C.

1. The function is defined and used in the same file
   (a) The definition lies above the function call. Example

```
double f(a); int a; { ... }   /* definition  */
double d;
d=f(2);                       /* call        */
```

   (b) The definition lies below the function call. In this case, the function must be declared above the function call. Example:

```
double d, f();                /* declaration */
d=f(2);                       /* call        */
double f(a); int a; { ... }   /* definition  */
```

2. The function is defined in one file and used in another file.

   (a) The function is declared above the function call. Example:

   ```
   double d, f();                /* declaration */
   d=f(2);                       /* call        */
   ```

   (b) The function declaration may be accessed by using the directive #include above the function call. Example:

   ```
   /* Include the file file.h which contains the declaration of
    * function f. */
   #include "file.h"             /* include declaration */
   double d;
   d=f(2);                       /* call                */
   ```

   (c) The function is neither declared nor defined above the function call (not even via #include). The function is in this case assumed to be of type int. Example:

   ```
   double d;
   d=f(2);                       /* call, assume type int */
   ```

   This may cause an error if the function f is not of type int.

If the function is declared outside the file in which it is used, one should organize the source code as in item 2b (the case with the #include directive item) above. This will ensure that the compiler is informed about any changes in the function's type made by the author of the function, or differences due to platform. With ANSI C, the compiler is also informed about the types of the function arguments. Also refer to appendix A.6.

Here are some examples of functions which people easily forget to type-specify

- malloc (defined in `<malloc.h>` or `<stdlib.h>`, depending on operating system and C dialect). Refer to appendix A.4 (XITE include directives) on how to avoid the problem of remembering which header file to include on a particular platform.
- strcpy and other string routines (defined in `<strings.h>` or `<string.h>`, depending on operating system and C dialect)
- pow and other math routines (defined in `<math.h>`)

A.4 XITE include directives

To relieve the single programmer from needing to remember what header file to include on a given platform, a few constants are predefined in XITE. These constants are used to grab the correct header file. Additionally, on some platforms they declare functions which are not found in any system header files.

Refer to appendix A.4.1.

The table below shows which constants are defined, which functions they are to be used with and which header files they ordinarily represent (depending on machine platform).

XITE constants	Functions	Header files
XITE_ENDIAN_H	"byte order"	<endian.h>, <sys/endian.h> <sys/machine.h> or <machine/endian.h>
XITE_FCNTL_H	"file usage"	<fcntl.h> or <sys/fcntl.h>
XITE_FILE_H	"file usage"	<sys/file.h> or <sys/io.h>
XITE_FORK_H	"forking"	<unistd.h> and/or <vfork.h>
XITE_IO_H	"file usage"	<io.h> or <xite/dummy.h>
XITE_LIMITS_H	pixtyp limits	<limits.h> and <values.h>
XITE_MALLOC_H	malloc, calloc, ...	<malloc.h> or <stdlib.h>
XITE_MEMORY_H	memcpy, memchr, ...	<memory.h> or <string.h>
XITE_MKTEMP_H	mktemp	<stdlib.h> or <unistd.h>
XITE_RANDOM_H	srandom, random	<math.h> or <stdlib.h>
XITE_STAT_H	"file usage"	<sys/stat.h>
XITE_STDARG_H	"vararg" functions	<stdarg.h> or <varargs.h>
XITE_STDIO_H	printf, scanf etc.	<stdio.h>
XITE_STRING_H	strcat, strlen, ...	<string.h> or <strings.h>
XITE_STRTOL_H	strtod, strtol	<stdlib.h>
XITE_TIME_H	clock, time	<time.h>
XITE_TOUPPER_H	tolower, toupper	<ctype.h>
XITE_TYPES_H	"type definitions"	<sys/types.h>
XITE_UNISTD_H		<unistd.h>

The constants are defined only if `<xite/includes.h>` (refer to subsection 6.1) is included. We recommend inclusion of `<xite/includes.h>` prior to any other XITE header files. The exact meaning of using the above constants, can be found by reading the files `<xite/includes.h>` and `<xite/xite_*.h>`.

A.4.1 Examples

An example is if you would like to use the function malloc. You should write the directive

```
#include XITE_MALLOC_H
```

in your program. On some platforms, this will result in the inclusion of `<stdlib.h>`, on others `<malloc.h>` will be included.

As a second example, if you would like to use the function toupper, you write

```
#include XITE_TOUPPER_H
```

On some platforms, `<ctype.h>` will be included, on other platforms no system header files are included, but tolower and toupper are declared in an XITE supplied header file.

As a third example, if you would like to use the function fprintf, you write

```
#include XITE_STDIO_H
```

This will include `<stdio.h>` on all present platforms, but also declare fprintf for some platforms which do not declare this function in `<stdio.h>` (e.g. traditional C on sun4/sparc under SunOS 4).

A.5 Warnings from the compiler

The C programming language is not fully defined. Several aspects are implementation dependent, i.e. different compilers may choose different solutions. One example mentioned above, in appendix A.2, concerned whether the type char is defined as 7 bits plus a sign or as 8 bits without a sign.

Some compilers are better than others at giving warnings about "dangerous" situations, as e.g. the use of syntax which may give problems on a different machine platform, or with a different compiler.

A compiler which behaves almost the same on all machine platforms, is the C compiler from gnu, gcc. We recommend that the programs are compiled with gcc and the option `-Wall` (equivalent to the combination of the options `-Wimplicit -Wreturn-type -Wformat -Wswitch -Wcomment`) as a check, regardless of what compiler is used during program development or to produce the final binary executable. Refer to `man gcc`.

The gnu compiler, gcc, should be used to test the ANSI code as well as the traditional C code. To compile the traditional code, add the compiler option `-UFUNCPROTO`. Additional options should be used to check for consistency in prototypes and between ANSI and traditional C code. The compilation script cxite (refer to section 2) will do this when given the option `-gw`.

In addition to gcc several machine platforms offer the possibility of using the program lint. This is not a compiler, but a syntax checker looking for "dangerous" code. Refer to `man lint`.

A.6 Organization of header files

All necessary prerequisites for using an external XITE function (i.e. the function type (in traditional C) or prototype (in ANSI C), in addition to required `typedef`s, macros and constants), should be declared in files which can be included with the directive

```
#include <xite/filename.h>
```

for some `filename.h`. We call these header files *global*.

Every source file which defines non static functions should include the header file which declares these functions. In this way the compiler will detect whether the declarations seen by other files match the definitions. Consider an example, where the file `biff.c` defines the function Imake_image, which is used by several other functions and main programs. The corresponding declaration is contained in `biff.h`, which is included by other source and header files. Assume that the definition of Imake_image in `biff.c` is changed in terms of function or parameter types. If we forget to update `biff.h` correspondingly, several other source and header files will be given incorrect information. However, if `biff.c` itself includes `biff.h`, the compiler will inform us about this kind of inconsistency, when `biff.c` is compiled.

No unnecessary files should reside in `<xite/...>`. Examples of unnecessary header files are those containing macros which are of no interest to anyone who only wants to call the XITE functions. The macros are perhaps only for internal use inside the functions. Other examples are local constants or small utility functions. These definitions should either be placed in the source file (and declared static if they are used only in one source file) or in one or more files local to the pertaining source directory (if they need to be shared), so that they can be included with the directive

```
#include "filename.h"
```

We call these header files *local*.

Files which reside globally as `<xite/...>` should be copies of files which are also found locally. In this way all relevant code can easily be moved, copied and packaged.

All routines which are declared in global include files should have a manual page description (otherwise they don't deserve to be located in a global header file).

Prior to adding your source code to XITE, i.e. prior to installing your header files in XITEs include directory, you may put them in a directory called `include/xite` below your own home directory for testing. When you compile your software, use the option `-I$HOME/include`, to enable the compiler to find your header files. After your software is contributed to XITE, you don't need to use this option.

References

[1] Svein Bøe. XITE – X-based Image Processing Tools and Environment – Programmer's Manual for version 3.4. Report 92, Image Processing Laboratory, Department of Informatics, University of Oslo, P. O. Box 1080 Blindern, 0316 Oslo, Norway, June 1998, http://www.mn.uio.no/ifi/english/research/groups/dsb/resources/software/xite/ProgrammersManual

[2] Svein Bøe. XITE – X-based Image Processing Tools and Environment – Reference Manual for version 3.47. Technical report, Image Processing Laboratory, Department of Informatics, University of Oslo, P. O. Box 1080 Blindern, 0316 Oslo, Norway, September 2004, http://www.mn.uio.no/ifi/english/research/groups/dsb/resources/software/xite/ReferenceManual/C
1, 8

[3] Svein Bøe. XITE – X-based Image Processing Tools and Environment – System Administrator's Manual for version 3.47. Report 91, Image Processing Laboratory, Department of Informatics, University of Oslo, P. O. Box 1080 Blindern, 0316 Oslo, Norway, June 2004, http://www.mn.uio.no/ifi/english/research/groups/dsb/resources/software/xite/SysAdmMan/.
1, 7

[4] Svein Bøe, Tor Lønnestad, and Otto Milvang. XITE – X-based Image Processing Tools and Environment – User's Manual for version 3.47. Report 56, Image Processing Laboratory, Department of Informatics, University of Oslo, P. O. Box 1080 Blindern, 0316 Oslo, Norway, June 2004, http://www.mn.uio.no/ifi/english/research/groups/dsb/resources/software/xite/UsersManual/.
1, 2, 7

[5] Tor Lønnestad. The BIFF Image Concept, File Format, and Routine Library. Report 29, Image Processing Laboratory, Department of Informatics, University of Oslo, P. O. Box 1080 Blindern, 0316 Oslo, Norway, February 1990. 7

REPORTS FROM THE IMAGE PROCESSING LABORATORY

79 Ingvil Hovig :
"Verktøy og metoder for komprimering av MR bilder"
Dr.Scient. (Ph.D) thesis August 1995.

80 Øyvind Akerhaugen :
"Automatisk plassering av navn på kart ved hjelp av simulated annealing"
Cand.Scient. (Master) thesis February 1996.

81 Marius Midtvik :
"Reversibel komprimering av MR bilder basert på statisk kildemodellering"
Cand.Scient. (Master) thesis May 1996.

82 Tor Øyvind Didriksen :
"Linjefinning og klassifikasjon med Random Hough-transform
— en eksperimentell studie"
Cand.Scient. (Master) thesis June 1996.

83 Edward Allen Smith :
"Image Processing Techniques on DNA Fingerprint Images
and its Application to Genetic Similarity Analysis"
Cand.Scient. (Master) thesis August 1996.

84 Parviz Heydari :
"Line Following in Digitized Map"
Cand.Scient. (Master) thesis November 1996.

85 Ramin Gordjianfar :
"Vectorization of the Cartographic Data"
Cand.Scient. (Master) thesis November 1996.

86 Luren Yang and Torfinn Taxt :
"Robust Methods for Sonar Bottom Detection"
February 1997.

87 Christian Wladimir Hansson :
"Strukturgrammatikk — en høyeredimensjonal grammatikk
for syntaktisk mønstergjenkjenning av 3D objekter i bilder"
Cand.Scient. (Master) thesis December 1996.

88 Sverre H. Huseby :
"Video on the World Wide Web —
Accessing Video from WWW Browsers"
Cand.Scient. (Master) thesis February 1997.

89 Håvard Lauritzen :
"Raster til vektor konvertering ved simulert størkning"
Cand.Scient. (Master) thesis May 1997.

90 Irene Rødsten :
"Texture Segmnetation using Moment based Features
obtained by Locally Adaptive Thresholding"
Cand.Scient. (Master) thesis May 1997.

91 Svein Bøe :
"XITE System Administrator's Manual"
August 1997.