Chapter 1

C Cross Referencing & Documenting tool. Version 1.6 - cxref

A program that can automatically generate documentation and cross references for a C program. The input is any C program with appropriate comments and the output is LaTeX, HTML, RTF or SGML files.

1.1 Program Options

The name of the program is cxref.

Usage: `cxref`

```
filename [ ... filename]
[-Odirname] [-Nbasename] [-Rdirname]
[-all-comments] [-no-comments]
[-verbatim-comments] [-block-comments]
[-xref[-all][-file][-func][-var][-type]]
[-warn[-all][-comment][-xref]]
[-index[-all][-file][-func][-var][-type]]
[-raw]
[-latex]
[-html[-src]]
[-rtf]
[-sgml]
[-Idirname] [-Ddefine] [-Udefine]
[-CPP cpp_program] [-- cpp_arg [ ... cpp_arg]]
```

Usage: `cxref`

```
filename [ ... filename] -delete
[-Odirname] [-Nbasename] [-Rdirname]
```

`filename` The name of the file to document, any number of files may be documented at a time.

`-delete` The files named are to be deleted from the output directory and their entries in the cross reference database and main output files are to be removed.

`-Odirname` The name of a directory to use for the output LaTeXfiles and the location of the cross reference files that are created.

`-Nbasename` The name to use for the first part of the output and cross reference files instead of cxref, the file extensions remain the same.

`-Rdirname` When the source files are in more than one directory, set dirname to the name of the root directory of the source tree (use relative path if easier e.g. "-R../.."). This will then run cxref from that root directory and the "-Odirname" must be relative to that directory.

`-all-comments` In case you think that the existing comments might work, (see below for description of special comments). *[Danger! This option can produce weird results.]*

`-no-comments` Ignores all comments, useful if you just want the cross references and not the documentation.

`-verbatim-comments` When the comments that you have in the code are formatted in a predetermined style that you want to preserve on the output, this option will force them not to be reformatted. *[Note, this is for file and function comments only.]*

`-block-comments` When the comments in the program are formatted in the "block" style (with a leading "*" character on every line), this option will remove that character from the output. *[Works for a single "*", "+", "—" or ":" on each line.]*

`-xref` Produce cross referencing information (see below).

- `-all` All cross references.
- `-file` Cross references for files.
- `-func` Cross references for functions.
- `-var` Cross references for variables.
- `-type` Cross references for types.

`-warn` Produce warnings, the options must be concatenated together:

- `-all` All warnings.
- `-comment` Warn of missing comments.
- `-xref` Warn of missing cross references.

`-index` Produce a cross reference index, the options must be concatenated together:

- `-all` All indexes.
- `-file` Index of files.
- `-func` Index of functions.
- `-var` Index of variables.
- `-type` Index of types.

`-raw` Produce a raw form of output, not really of much use except with -warn.

`-latex` Produce a LaTeX file to document each of the source files and also an extra file that includes each of these files.

`-html` Produce an HTML 4.01 file to document each of the source files and a main file to reference each of these files.

`-html-src` Produce the HTML output and an HTML version of the source file with links into it.

`-rtf` Produce a Rich Text Format (RTF) file to document the source file.

`-sgml` Produce an SGML file to document the source file. (Using the LinuxDoc DTD).

`-Idirname` GCC option to specify the path for include files.

`-Ddefine` GCC option to define a pre-processor symbol.

`-Udefine` GCC option to undefine a pre-processor symbol.

`-CPP program` The name of the program to use instead of the compile time default. The program must be able to perform all of the actions that "gcc -E -C -dD" does to work. If the program takes arguments then the whole thing needs to be in quotes so that it is interpreted as a single argument to cxref.

`-- arg ... arg` Extra arguments to be passed to the pre-processor can be placed after the "–" separator.

1.2 C Compiler Replacement cxref-cc

To simplify using cxref on existing source code, there is now a shell script that will call the C compiler and then call cxref to process the source file. This means that it can be used as a drop in replacement for CC in Makefiles and the like.
Usage: cxref-cc [usual cc options]
The name of the source file is extracted from the list of options as well as the "-D*", "-I*", "-U*" flags and when the C compiler exits succesfully cxref will be called. The name of the C compiler to use is controlled by the CXREFCC environment variable, or if this is not set then the CC environment variable, or failing this just gcc.
Using this script requires the use of a '.cxref' configuration file to contain the options since there is nowhere to put the options on the command line for the C compiler.
This will only cross-reference and document the C source files since they are the only ones that are compiled, but it will make sure that they are cross-referenced with the correct options etc.

1.3 Cxref configuration File

These command line arguments can also be put into a file named ".cxref" instead of on the command line. When cxref is run the arguments to the program are interpreted in the following order.

1. Those on the command line.
2. Those in the ".cxref" file in the current directory.
3. Those in the ".cxref" file in the source tree root specified by "-R".

This means that in a multi-directory source tree, each sub-directory can have a ".cxref" file containing just the line "-R.." or appropriate. The main directory can have a ".cxref" file containing the remainder of the options. This removes completely the need to have any options on the command line apart from the source file names.
The format of the .cxref file is any number of lines, each one containing a single command line argument (equivalent to one of the argv). The only options that cannot be used are the names of source files themselves and the "-delete" option. Blank lines are ignored and lines starting with a '#' are comments.

1.4 Program Documentation Comments

The documentation for the program is produced from comments in the code that are appropriately formatted. The cross referencing comes from the code itself and requires no extra work.
The special comments are "/**** ****/" (for a file) and "/*+++++ +++++*/" (for a data object) any number of "*" or "+" can be used inside of the standard "/*" and "*/" comment delimiters in the comments, they are ignored.
If a comment line starts with whitespace and is followed by '+html+' then the rest of the line is included only in the HTML output, and is not processed so it can include HTML markup, '-html-' means that the rest of the line is included in all except the HTML output. The same also applies to the other output formats, '+none+' can be used for lines not to appear in any output. The exception to this is that the raw output does not do any checking and will output all lines.
In any situation where a comment follows a ",", ";" or ")" separated only by spaces and tabs, the comment is pushed to before the punctuation to apply to the object there.
The program is implemented using a full ANSI C grammar parser with some GCC extensions, this means that the style of the code is unimportant, only the content and comments.

1.5 Automated Comment Insertion

To simplify the insertion of comments that will be parsed by cxref, the file cxref.el provides a number of Emacs lisp functions. To use them add the line `(load "cxref")` to your ".emacs" file or type `M-x load-file cxref.el` from within Emacs.
The functions and key bindings are:

Control-C Control-F Adds file comments, a /** **/ header at the top of the file and if it is a .h file then it also adds a #ifndef, #define at the beginning and #endif at the end to stop multiple inclusions.

Control-C f Adds comments to a function, the cursor must be on the line containing the start of the function definition when this function is called. The /*+ ... +*/ comment that is added is of the header type (see the examples) not inline.

Control-C v Adds a leading comment to the variable or other definition on the current line.

Control-C e Adds a trailing comment at the end of the line.

Control-C i Adds an inline comment that is ignored by cxref.

1.6 C Preprocessor

To improve the output that is available a modified version of the GNU CPP V2.7.2 is supplied (named cxref-cpp).
This modified C preprocessor allows for a finer control over some features of the preprocessing that are not important for a compiler. In a standard preprocessor, the preprocessor directives are intended for use only by the preprocessor, so passing the information through is not important.
With cxref-cpp, there are two features that are different to the standard GNU CPP:

1. The #include directives from the file are output in the same way as the #defines are output. An extra flag has been added to cpp to do this, '-dI', it works in the same way as the existing '-dD' flag for #defines.
2. Comments trailing a #include or a #define are dropped with GNU CPP even if -C is used. This is not important while compiling but is useful for documenting.

1.7 Cross Referencing

The cross referencing is performed for the following items

Files
- The files that the current file is included in (even when included via other files).

#includes
- Files included in the current file.
- Files included by these files etc.

Variables
- The location of the definition of external variables.
- The files that have visibility of global variables.
- The files / functions that use the variable.

Functions
- The file that the function is prototyped in.
- The functions that the function calls.
- The functions that call the function.
- The files and functions that reference the function.
- The variables that are used in the function.

Each of these items is cross referenced in the output.
The cross referencing uses files 'cxref.variable', 'cxref.function', 'cxref.include' and 'cxref.typedef' in the output directory.
These are a complete list of the function and variable usage in the program and could be used to generate a function call hierarchy or variable usage diagram for example.
Two cxref passes of each file is needed, the first to build up the cross referencing files and the second to use them.
(The file names are different if the "-N" option is used.)

1.8 LaTeXOutput

The default LaTeXoutput is a file for each of the source files with one extra file "cxref.tex" that includes each of the other files. This is to allow a makefile to only update the changed files (although the references may require all of the files to be checked again). When the cxref.tex file has been written it can be modified by the user, any new files that are added are added at the end of the source code section, the rest of the file being unchanged.
The index is written to a file called "cxref.apdx.tex" and cxref.tex is updated to refer to it.
Also written out are three LaTeXstyle files "page.sty", "fonts.sty" and "cxref.sty". These set up the page to use a smaller margin and smaller fonts to allow more to appear on a page and also define the new commands for typesetting the cxref output.
(The file names "cxref.tex" and "cxref.apdx.tex" are different if the "-N" option is used.)

1.9 HTML Output

The default HTML output is a file for each of the source files with one extra file "cxref.html" that includes each of the other files. This is to allow a makefile to only update the changed files (although the references may require all of the files to be checked again). When the cxref.html file has been written it can be modified by the user, any new files that are added are added at the end before the table of contents, the rest of the file being unchanged.
The index is written to a file called "cxref.apdx.html" and cxref.html is updated to refer to it.
(The file names "cxref.html" and "cxref.apdx.html" are different if the "-N" option is used.)
The HTML output uses the HTML 4.01 standard and all pages validate correctly.

1.10 RTF Output

Rich Text Format is a fairly low level page description format devised by Microsoft. It is not a well defined and easy to understand standard as are the other formats, but it is popular for document exchange.
There is a single output file for each of the source files and an index file.

1.11 SGML Output

Since SGML is a meta-language it is necessary to define the layout elements as well as provide the information. The cxref output uses the LinuxDoc document format and is designed for use with the SGMLtools programs (http://www.sgmltools.org/).
There is a single output file for each of the source files and an index file.

1.12 Example Special Comments

The file "README.c" is included at the end of this document to show that the comments are indeed seen in the code, the result of running this through cxref is also included.

1.13 Further Information

There is a list of frequently asked questions and their answers for the cxref program in the FAQ file. A list of improvements planned for future versions of the program are listed in the file TODO.
More up-to-date information can be found on the World Wide Web at the cxref homepage, reached via the author's homepage http://www.gedanken.demon.co.uk/.
If you wish to submit bug reports or other comments about the program then email the author amb@gedanken.demon.co.uk and put cxref in the subject line.

1.14 Author and Copyright

The cxref program was written by Andrew M. Bishop in 1995-2004.
The cxref program is copyright Andrew M. Bishop 1995-2004.
The cxref-cpp program is copyright Free Software Foundation, Inc.

Chapter 2

Example "README.c"

```
/*****************
  $Header: /home/amb/cxref/doc/RCS/README.c 1.4 1997/05/26 11:23:40 amb Exp $

  A comment for the file, RCS header comments are treated specially when first.
  *****************/



/*+ A #include comment +*/
#include <stdio.h>


#include <math.h>  /*+ An alternative #include comment. +*/


/*+ A #define comment. +*/
#define def1 1


#define def2 2  /*+ An alternative #define comment. +*/


/*++++++++++++
  A #define with args

  arg1 The first arg

  arg2 The second arg
  ++++++++++++*/

#define def3(arg1,arg2) (arg1+arg2)


/*+ An alternative #define with args. +*/

#define def4(arg1 /*+ The first arg  +*/, \
             arg2 /*+ The second arg +*/) \
           (arg1+arg2)


/*+ An example typedef comment +*/
typedef enum
{
 one,            /*+ one value +*/
```

```
 two            /*+ another value +*/
}
type1;


/*+ Another example typedef comment, +*/
typedef struct
{
 int a;         /*+ A variable in a struct. +*/
 union bar
  {
   char a;      /*+ Each element +*/
   int  b,      /*+ of a struct +*/
        c;      /*+ or a union +*/
   long d;      /*+ can have a comment +*/
  }
 e;             /*+ Nested structs and unions also work. +*/
}
type2,          /*+ a type that is a struct. +*/
*ptype2;        /*+ a pointer to a struct type. +*/


/*+ A leading comment only. +*/
int var1,var2;


static int var3; /*+ A trailing comment only. +*/


/*+ A variable for +*/
int var4,    /*+ one thing. +*/
    var5,    /*+ a second thing. +*/
    var6;    /*+ a third thing. +*/

/* Note: The leading comment is combined with each of the trailing comments. */
/* Note: the push through of the comment above on the ',' and ';', see README. */


/*++++++++++++
  A function comment (the comments for the args need to be separated by a blank line).

  int function1 The return value.

  int arg1 The first argument.

  int arg2 The second argument.

  Some more comments

  +none+  Note: this line and the two below are processed specially.
  +html+  This comment is only visible in the HTML output, and can contain HTML markup.
  +latex+ This comment is only visible in the \LaTeX output, and can contain \LaTeX markup.
  ++++++++++++*/

int function1(int arg1,int arg2)
{
 /*+ An internal comment in a function that appears as a
   new paragraph at the end of the comment. +*/
```

```
 var1=0;

 function2(var3,var4);
}


/*+ An alternative function comment +*/

int function2(int arg1,  /*+ The first argument.  +*/
              int arg2)  /*+ The second argument. +*/
/*+ Returns a value +*/
{
 int (*funcp)()=&function1;
}

/* Note: the push through of the comment above on the ',' and ')', see README. */
```

Chapter 3

Cxref Output For "README.c"

3.1 File README.c

RCS Header: /home/amb/cxref/doc/RCS/README.c 1.4 1997/05/26 11:23:40 amb Exp

A comment for the file, RCS header comments are treated specially when first.

Included Files

A #include comment

`#include <stdio.h>`

An alternative #include comment.

`#include <math.h>`

Preprocessor definitions

A #define comment.

`#define def1 1`

An alternative #define comment.

`#define def2 2`

A #define with args

`#define def3( arg1, arg2 )`

· `arg1`	The first arg
· `arg2`	The second arg

An alternative #define with args.

`#define def4( arg1, arg2 )`

· `arg1`	The first arg
· `arg2`	The second arg

3.1.1 Type definitions

3.1.1.1 Typedef type1

An example typedef comment

`typedef enum {...} type1`

```
enum
  {
    one;                    one value
    two;                    another value
  }
```

3.1.1.2 Type union bar

Nested structs and unions also work.

```
union bar
  {
    char a;                      Each element
    int b;                       of a struct
    int c;                       or a union
    long d;                      can have a comment
  }
```

3.1.1.3 Typedef type2

Another example typedef comment, a type that is a struct.

```
typedef struct {...} type2
```

```
struct
  {
    int a;                       A variable in a struct.
    union bar
      {
        char a;                  Each element
        int b;                   of a struct
        int c;                   or a union
        long d;                  can have a comment
      }
      e;                         Nested structs and unions also work.
  }
```

3.1.1.4 Typedef ptype2

Another example typedef comment, a pointer to a struct type.

```
typedef struct {...}* ptype2
```

See: Typedef type2 *(Section 3.1.1.3)*

3.1.2 Variables

3.1.2.1 Variable var1

A leading comment only.

```
int var1
```

Visible in: README.c *(Section 3.1)*
Used in: function1() README.c *(Section 3.1.3.1)*

3.1.2.2 Variable var2

A leading comment only.

```
int var2
```

Visible in: README.c *(Section 3.1)*

3.1.2.3 Variable var4

A variable for one thing.

```
int var4
```

Visible in: README.c *(Section 3.1)*
Used in: function1() README.c *(Section 3.1.3.1)*

3.1.2.4 Variable var5

A variable for a second thing.

`int var5`

Visible in: README.c *(Section 3.1)*

3.1.2.5 Variable var6

A variable for a third thing.

`int var6`

Visible in: README.c *(Section 3.1)*

3.1.2.6 Local Variables

var3
A trailing comment only.

`static int var3`

Used in: function1() *(Section 3.1.3.1)*

3.1.3 Functions

3.1.3.1 Global Function function1()

A function comment (the comments for the args need to be separated by a blank line).

`int function1 ( int arg1, int arg2 )`

· `int function1`	The return value.
· `int arg1`	The first argument.
· `int arg2`	The second argument.

Some more comments
This comment is only visible in the LaTeXoutput, and can contain LaTeXmarkup.
An internal comment in a function that appears as a new paragraph at the end of the comment.

Calls:	function2()	README.c	*(Section 3.1.3.2)*
Used in:	function2()	README.c	*(Section 3.1.3.2)*
Refs Var:	var1	README.c	*(Section 3.1.2.1)*
	var3	README.c	*(Section 3.1.2.6)*
	var4	README.c	*(Section 3.1.2.3)*

3.1.3.2 Global Function function2()

An alternative function comment

`int function2 ( int arg1, int arg2, void )`

· `int function2`	Returns a value
· `int arg1`	The first argument.
· `int arg2`	The second argument.
· `void`	

Called by:	function1()	README.c	*(Section 3.1.3.1)*
Refs Func:	function1()	README.c	*(Section 3.1.3.1)*

Contents