A C++ Browser for GNU Emacs

Gerd Moellmann

1 Distribution

The C++ browser is "free"; this means that everyone is free to use it and free to redistribute it on a free basis. The browser is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of the C++ browser that they might get from you. The precise conditions are found in the GNU Emacs General Public License that comes with Emacs and also appears following this section.

The easiest way to get a copy of the browser is from someone else who has it. You need not ask for permission to do so, or tell any one else; just copy it.

If you have access to the Internet, you can get the latest distribution version of GNU Emacs from host 'prep.ai.mit.edu' using anonymous login. See the file '/u2/emacs/GETTING.GNU.SOFTWARE' on that host to find out about your options for copying and which files to use.

You may also receive GNU Emacs when you buy a computer. Computer manufacturers are free to distribute copies on the same terms that apply to everyone else. These terms require them to give you the full sources, including whatever changes they may have made, and to permit you to redistribute the GNU Emacs received from them under the usual terms of the General Public License. In other words, the program must be free for you when you get it, not just free for the manufacturer.

If you cannot get a copy in any of those ways, you can order one from the Free Software Foundation. Though Emacs itself is free, our distribution service is not. An order form is included at the end of manuals printed by the Foundation. It is also included in the file 'etc/DISTRIB' in the Emacs distribution. For further information, write to

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
USA

The income from distribution fees goes to support the foundation's purpose: the development of more free software to distribute just like GNU Emacs.

If you find GNU Emacs useful, please send a donation to the Free Software Foundation. This will help support development of the rest of the GNU system, and other useful software beyond that. Your donation is tax deductible.

2 GNU Emacs General Public License

(Clarified 11 Feb 1988)

The license agreements of most software companies keep you at the mercy of those companies. By contrast, our general public license is intended to give everyone the right to share GNU Emacs. To make sure that you get the rights we want you to have, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. Hence this license agreement.

Specifically, we want to make sure that you have the right to give away copies of Emacs, that you receive source code or else can get it if you want it, that you can change Emacs or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of Emacs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for GNU Emacs. If Emacs is modified by someone else and passed on, we want its recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

Therefore we (Richard Stallman and the Free Software Foundation, Inc.) make the following terms which say what you must do to be allowed to distribute or change GNU Emacs.

2.1 Copying Policies

1. You may copy and distribute verbatim copies of GNU Emacs source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each file a valid copyright notice "Copyright (C) 1988 Free Software Foundation, Inc." (or with whatever year is appropriate); keep intact the notices on all files that refer to this License Agreement and to the absence of any warranty; and give any other recipients of the GNU Emacs program a copy of this License Agreement along with the program. You may charge a distribution fee for the physical act of transferring a copy.
2. You may modify your copy or copies of GNU Emacs source code or any portion of it, and copy and distribute such modifications under the terms of Paragraph 1 above, provided that you also do the following:
   - cause the modified files to carry prominent notices stating who last changed such files and the date of any change; and
   - cause the whole of any work that you distribute or publish, that in whole or in part contains or is a derivative of GNU Emacs or any part thereof, to be licensed at no charge to all third parties on terms identical to those contained in this License Agreement (except that you may choose to grant more extensive warranty protection to some or all third parties, at your option).
   - if the modified program serves as a debugger, cause it, when started running in the simplest and usual way, to print an announcement including a valid copyright notice "Copyright (C) 1988 Free Software Foundation, Inc." (or with the year that

is appropriate), saying that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License Agreement.

- You may charge a distribution fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Mere aggregation of another unrelated program with this program (or its derivative) on a volume of a storage or distribution medium does not bring the other program under the scope of these terms.

3. You may copy and distribute GNU Emacs (or a portion or derivative of it, under Paragraph 2) in object code or executable form under the terms of Paragraphs 1 and 2 above provided that you also do one of the following:
   - accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Paragraphs 1 and 2 above; or,
   - accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal shipping charge) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Paragraphs 1 and 2 above; or,
   - accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form alone.)

   For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are standard libraries that accompany the operating system on which the executable file runs.
4. You may not copy, sublicense, distribute or transfer GNU Emacs except as expressly provided under this License Agreement. Any attempt otherwise to copy, sublicense, distribute or transfer GNU Emacs is void and your rights to use GNU Emacs under this License agreement shall be automatically terminated. However, parties who have received computer software programs from you with this License Agreement will not have their licenses terminated so long as such parties remain in full compliance.
5. If you wish to incorporate parts of GNU Emacs into other free programs whose distribution conditions are different, write to the Free Software Foundation. We have not yet worked out a simple rule that can be stated here, but we will often permit this. We will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software.

Your comments and suggestions about our licensing policies and our software are welcome! Please contact the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139.

2.2 NO WARRANTY

BECAUSE GNU EMACS IS LICENSED FREE OF CHARGE, WE PROVIDE ABSOLUTELY NO WARRANTY, TO THE EXTENT PERMITTED BY APPLICABLE

STATE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING, FREE SOFTWARE FOUNDATION, INC, RICHARD M. STALLMAN AND/OR OTHER PARTIES PROVIDE GNU EMACS "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE GNU EMACS PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW WILL FREE SOFTWARE FOUNDATION, INC., RICHARD M. STALLMAN, AND/OR ANY OTHER PARTY WHO MAY MODIFY AND REDISTRIBUTE GNU EMACS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH PROGRAMS NOT DISTRIBUTED BY FREE SOFTWARE FOUNDATION, INC.) THE PROGRAM, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

3 Introduction

I am myself working in C++ commercial software projects for several years now, and in these projects I always missed software support for two things:

- When a new class library is introduced or you have to work on some source code you haven't written yourself (or written sufficiently long ago) you need a tool letting you navigate through class hierarchies and investigate features of the software. Without such a tool you often end up grep'ing through dozens or even hundreds of files.
- Once you are productive, it would be nice to have a tool that knows your sources and can help you while you are editing source code. Imagine to be able to jump to the definition of an identifier while you are editing, or something that can complete long identifier names because it knows what identifiers are defined in your program. . . .

The design of the *C++ class browser* reflects these two needs; therefore the name *class browser* is really only half the truth.

How does it work?

A fast parser, and I mean fast, for a subset of C++ written in C is used to parse C++ source and header files. The parser generates a database containing information about identifiers—classes, members, global functions, types etc.

The second part of the C++ browser is a Lisp program comprised of a number of Lisp source files—those with names `br-*.el`. This package reads the database generated by the parser; it displays its contents in various forms and allows you to perform operations on it, or with the help of the knowledge contained in it.

Navigational use of the browser is centered around two main buffer types which define their own major modes:

Tree buffers are used to view class hierarchies in tree form. They allow you to quickly find classes, find or view a class declaration, perform operations like query replace on sets of files containing classes, and finally they are used to produce the second buffer form: member displays.

Members are displayed in *member buffers*. The browser distinguishes between six different types of members; each type is displayed as a member list of its own:

- Instance member variables,
- Instance member functions,
- Static member variables,
- Static member functions,
- Friends,
- Types (`enum`s, and `typedef`s defined with class scope).

You can quickly switch member buffers from one list to another, or to another class, you can include inherited members in a display, you can set filters, and most important of all you can edit or view member declarations and definitions with a keystroke.

These two buffer types and the commands they provide support the *navigational* use of the browser. The second form resembles the *tags* package for C and some other procedural languages that is part of the standard Emacs distribution; its commands are not bound to special buffer types but are most often used while you are editing your source code.

To list just some of what you can use the *tags* part of the browser for:

- Quickly jump to the definition or declaration of an identifier in your source code.
- Complete identifiers in your source—just type in a few characters and let Emacs fill in the rest.
- Perform search and query replace operations over some or all of your source files.
- Show all identifiers matching a regular expression—and jump to one of them, if you like.

The browser has already been used in projects containing hundreds of classes. The maximum project size that has been successfully tried so long was about 500,000 lines of code. With Emacs 18.55 it runs on a `MS-DOS` notebook with 4 Mb of RAM and has no problems to handle projects up to about 70,000 lines of code.

3.1 Current Program Versions

The version number of the browser described in this manual is 3.1. This version is the first released on the Internet. Version 1.x has been published on Compuserve, versions 2.x have never seen the light of the public due to its lack of documentation.

Beginning with this version of the package, support for Emacs 18 will begin to be dropped because there is now an Emacs version 19.27 which runs on MS-DOS (thanks to Eberhard Mattes for his OS/2 port).

3.2 Author

These programs were written by

Gerd Moellmann, Duesseldorf Germany
mmann@ibm.net
Compuserve: 100025,3303

Suggestions, comments, bug reports, and requests for improvements are welcome.

4 Installation

The following sections describe how the browser package can be installed on your system. At the time of writing, the package has been used with Emacs 18.55 and 19.19 through 19.27 on DOS and OS/2. With this version I am beginning to drop support for Emacs 18 but I kept the installation instructions in the document this time.

Unfortunately I currently do not have access to a Unix system, so it hasn't been tested on these platforms yet. From my knowledge of Unix and Emacs running under Unix I think there should barely be any problems.

4.1 Distribution Contents

The following files are part of the browser distribution. All files are needed to successfully use the browser; please check that you got them all.

`lex.h, lex.c, parse.c, sym.c`
: These C source files contain the parser `ebrowse` that generates the Lisp database. They are written in *K&R* C so that they should compile with pre-ANSI compilers. There is no platform specific code in the parser except for special handling of carriage return characters under OS/2 and DOS (that might come handy under VMS). The code is 32-bit clean; it has been used under MS-DOS and 32-bit OS/2 with various C and C++ compilers (Borland, IBM, Gnu).

`makefile`
: A DMAKE v3.8 makefile for the OS/2 version of `ebrowse`. This makefile is not essential. The compilation of the C source files is so easy that you can easily type in the single command.

`br-brows.el`
`br-add.el`
`br-compl.el`
`br-loop.el`
`br-macro.el`
`br-membe.el`
`br-menu.el`
`br-posit.el`
`br-save.el`
`br-stat.el`
`br-struc.el`
`br-tags.el`
`br-tree.el`
`br-trees.el`
`br-utils.el`
`br-rev.el`
: Emacs Lisp source files that together form the Lisp part of the browser. These files are compatible with Emacs 18 and Emacs 19 (I am now dropping the support for Emacs 18 because there is an Emacs 19.27 running under DOS—i.e. on my notebook computer;-))

If you are using Emacs 18, two other Lisp packages are needed to run the browser; for convenience I have included them in the browser package, they are described below.

`cl-compa.el`
`cl-extra.el`
`cl-macs.el`
`cl-seq.el`
`cl.el` This is the unchanged `CL-19` package from Emacs 19.19. If you are not using Emacs 18, you should have these files already.

`byte-opt.el`
`byte-run.el`
`bytecomp.el`
This is the optimizing byte compiler that came with Emacs 19.19. Unfortunately, RMS removed a lot of code from the original 19.19 `bytecomp.el` that made it compatible with Emacs 18. Fortunately he didn't delete the code (god bless his soul) but commented it out instead, so I could reconstruct a version of the compiler that works with Emacs 18.

These files are MS-DOS only. If you already have a working optimizing compiler, or if you are running Emacs 19, you don't need them.

The optimizing compiler must be installed with Emacs 18 because the original compiler has a number of flaws that make it unusable with the CL-19 Common Lisp compatibility package. Besides that the new compiler produces much better code than the old one so there is no reason to stick with it.

`browser.texi`
This file is the Texinfo source code of the file you are currently reading. It contains the documentation of the browser package. This file is for Texinfo 3.0. Make sure that you have that package if you are using Emacs 18.

4.2 Building and Installing `ebrowse`

In order to build `ebrowse`, first of all make sure that you have all files of the parser (see also See Section 4.1 [Files], page 9, for a description of what is part of the distribution).

On Unix and OS/2 the following simple command should do the job of compiling `ebrowse`:

```
gcc -O2 -DPROTOTYPES lex.c parse.c sym.c -o ebrowse
```

Use an equivalent command line for your compiler if you don't have the GNU C compiler.

4.3 Installing the Lisp Source

The installation of the Lisp source differs between Emacs version 18 and Emacs 19. The reason for this is that the Emacs 18 byte compiler and its Common Lisp compatibilty package `CL` both have to be replaced.

If you are an Emacs 19 user, skip the first two sections.

4.3.1 Emacs 18 Byte Compiler Patch

The standard Emacs 18 byte compiler, located in `bytecomp.el`, has problems compiling some forms generated by the `CL-19` package. `CL-19` already applies one patch to the byte compiler (related to macro expansion of top-level forms), but this is not sufficient as I had to find out.

This package therefore includes the new optimizing byte compiler written by Jamie Zawinski and Hallvard Furuseth which in its original form ran both under Emacs 18 and 19. Since I only had access to a modified source that was part of the Emacs 19.19 distribution I had partly to port it back to Emacs 18. If you have the original source of the optimizing byte compiler running under Emacs 18, do not install the files in the browser package.

To install the compiler, save your original `bytecomp.el` (usually located in some directory like `/usr/local/emacs/lisp` in some backup directory, copy the new files `byte-opt.el`, `byte-run.el` and `bytecomp.el` to where it was located, start your Emacs and compile the files with *M-x byte-compile-file*. Do that again after loading `bytecomp.elc` with *M-x load-library* to ensure that the optimizing compiler has generated optimized code for itself.@refill

4.3.2 Installing the CL-19 Package

The `CL-19` package included in the `browse` package is the one that was part of Emacs 19.19. This package has a bug in the function `add-hook` that is fixed in `br-fixes.el` without modifying the original source. (The `CL-19` package defines this function for the benefit of Emacs 18 users that want to use the convenient Emacs 19 `add-hook`.)

Save the original `cl.el` in some backup directory. Then copy the Lisp source files `cl.el`, `cl-extra.el`, `cl-macros.el`, `cl-compat.el`, and `cl-seq.el` to your Emacs lisp directory and compile the files via *M-x byte-compile-file*.

4.3.3 Installing the other Lisp Files

After installing the optimizing byte compiler for Emacs 18 and the CL-19 package, the browser Lisp files (`br-*.el` can be compiled. I recommend not to copy the files into your Emacs Lisp directory. Choose some other directory to make upgrading both the browser and Emacs easier.

First compile the files `br-macros.el` and `br-structs.el` via *M-x byte-compile-file*; then load both compiled files `br-macros.elc` and `br-structs.elc` with *M-x load-library*. As a final step compile the rest of the files prefixed with `br-`.

4.4 Installing the Texinfo file

There are two ways to generate the hypertext documentation from the input file `browser.texi`. The preferred one used the command `makeinfo` as shown below:

```
makeinfo browser.texi
```

This generates the file `browser.info` in the current directory which can then be copied to where you keep your other info files.

The second method uses the internal Texinfo compiler: Load the file `browse.inf` into an Emacs buffer and run the command *M-x texinfo-format-buffer*. Save the resulting

file in the same directory where your other Texinfo files are kept (usually something like `/usr/local/emacs/info`).

As a last step, add a line like

```
* C++ Browser: (browser). The C++ browser...
```

at the end of the file `dir` found there. The file name `browser` following the colon above has to be replaced with the name under which you saved the result of *`M-x texinfo-format-buffer`*.

4.5 Modifying your Startup File

In order the be able to run the C++ browser, the following line has to be added to your `.emacs` (`_emacs` for the MS-DOS version of Emacs):

```
(require 'browse "br-brows")
```

Under MS-DOS/ Emacs 18 you should include the following lines in your `_emacs` in front of the above line. This makes sure that a CL-19 fix is installed before CL-19 is loaded.

```
(require 'br-fixes)
(require 'cl-19 "cl")
```

You might also want to bind some browser commands to global keys. The file `br-utils.el` binds keys prefixed with *`\C-c b`* to browser commands, and provides an electric buffer menu displaying tree buffers only. If you want to install these utilities, add to your `.emacs`:

```
(require 'br-utils)
```

When you are using Emacs 19 under a windowing system like `X11` or `PM`, you might find it usefull to have some browser commands available via the menu bar. If you want to do that, add to your `.emacs`:

```
(require 'br-menu)
```

See See Chapter 13 [Customization], page 37, for advanced customization of the behaviour of the browser program.

5 Processing Source Files

Before you can start browsing a class hierarchy you must run `ebrowse` (`ebrowse.exe` on MS-DOS or OS/2) on your source files to generate a Lisp database.

The operation of `ebrowse` can be tailored with command line switches described below. Under normal circumstances it suffices to let the parser use its default settings. If you want to do that, call it with a command line like:

```
ebrowse *.h *.cc
```

or, if your are using a response file (see below),

```
ebrowse -i files.list
```

`Ebrowse` prints a list of available command line switches when it is called with no arguments.

5.1 Specifying Input Files

'`file...`' Specify one or more files to parse.

Depending on the operating system on which `ebrowse` is used, you might be able to specify all input files to be parsed on the command line. On some systems (MS-DOS and OS/2), the maximum command line length is normally insufficient for the number of files in even small C++ programs. This is one reason for the introduction of response files described below.

'`-i file`' This command line switch specifies a response file containing a list of files to be parsed by `ebrowse`. Each line in the response file has to contain one file name. More than one option of this kind is allowed. The maximum number of response file options is set at compilation time of `ebrowse`. The default value is 50.

It is generally a good idea to form the input files list so that header files are parsed before source files are. Header files normally contain class declarations. This facilitates the parsers work of properly identifying friend functions of a class. If a global function is parsed before the parser knows that the function is a friend function of a class, this friend relationship information is currently lost.

5.2 Changing the Output File Name

'`-o file`' By default, `ebrowse` produces a data base named `BROWSE` in the directory in which it is called. You can change the output file name by giving a different name on the command line following the above command line switch.

'`-a`' By default, each run of `ebrowse` erases the old contents of the output file when writing to it. You can append output to an existing file by giving an '`-a`' anywhere on the command line.

5.3 Friend Function Processing

'`-f`' Friend functions are global functions that do not have a direct syntactic linkage to classes like member functions have. When seeing a global function, the parser therefore has to search its symbol table for a friend declaration matching the function just parsed.

You can suppress this search by giving the above command line option.

Normally you need not do so because the number of global functions in a C++ program tends to be quite small so that the overhead of the symbol table search is neglectable. If you are gradually converting large C programs to C++ so that the number of classes is initially small and the number of global functions is high, this option can prove usefull because it improves parsing speed.

5.4 Structs and Unions

'`-s`' This switch suppresses all classes declared as `struct` or `union` in the output.

This is for the case that you are converting an existing C program to C++, and do not want to see C structs in the tree.

5.5 Regular Expressions

By default, the parser only records a member's or class' name in the output file along with the file and the position in the file where its declaration or definition was found. The Lisp program part then builds suitable regular expressions to find a class or member in the file even if the file has been edited since the last time `ebrowse` was run on it. This works almost always fine if you have a *normal* coding (indentation) style.

For those cases where the regular expression guessing algorithms in the Lisp package prove to be too weak you can instruct the parser to produce regular expressions from what is found in the source files. These regular expressions can be stored either as part of the normal output file (for small to medium databases), or in a separate file (for big, fat databases). In the latter case, the file containing the regular expressions is loaded on an as needed basis; you are always free to remove its buffer to free memory.

'`-x`' This option turns regular expression production on.

'`-r file`' Instead of writing regular expressions to the file `BROWSE`, write them to `file`. The file will be loaded by the Lisp program when it is needed and can be discarded by the user when it is no longer used.

'`-m#`' The number following this option specifies the minimum length of the regular expressions produced to match class and member declarations and definitions. The default value is set at compilation time of `ebrowse`. The smaller the minimum length the higher the probability that the browser will find a wrong match. The larger the value the larger the output file and therefore the memory consumption once the file is read from Emacs.

'`-M#`' The number following this option specifies the maximum length of the regular expressions used to match class and member declarations and definitions. The default value is set at compilation time of `ebrowse`. The larger the maximum

length the higher the probability that the browser will find a correct match, but the larger the value the larger the output file and therefore the memory consumption once the data is read from Emacs. As a second effect, the larger the regular expression the higher the probability that it will no longer match after editing the file.

'`-v`' When this option is given on the command line, a '`.`' is displayed for each file parsed, and a '`+`' is displayed for each class written to the output file. A summary of the memory used by the parser is displayed when parsing is finished.

'`-V`' This option makes `ebrowse` print out the names of the files while it is parsing them and the names of the classes added to the symbol table.

'`-e<file>`'

This option is only used when `ebrowse` is called from the Lisp package. It lets `ebrowse` read from standard input and write to standard output, `file` is assumed to be the name of the file from which the input originally comes from—this name is written as part of the position information in the resulting Lisp database.

6 Starting to Browse

You start browsing a Lisp database produced by `ebrowse` by just *finding* the output file with `\C-x \C-f`. The browser installs an Emacs *find file hook* that checks every file loaded into memory for a certain magic number. If it finds a browser database, it is converted into internal form that is subsequently displayed in a tree buffer.

This way of loading a database integrates smoothly with other Emacs features: you can set bookmarks on browser databases, you can start browsing with `f` in *dired* mode, you can temporarily display a tree with `M-x view-file` etc.

An example of a tree buffer display is shown below.

```
|  Collection
|    IndexedCollection
|      Array
|        FixedArray
|    Set
|    Dictionary
```

When you run Emacs 19.27 or greater under a windowing system like X11 you will notice that that certain areas in the tree buffer are highlighted when you move the mouse over them. This highlight marks *sensitive regions* in the buffer. Clicking `mouse-2` on such a region will pop up an *object menu* depending on the type of region you are in. This menu contains functions that can be performed on the object that is highlighted.

Each buffer has a *buffer object menu* that is similarily opened with a click of `mouse-2` somewhere in the buffer where no highlight is displayed.

7 Tree Buffers

Class trees are displayed in *tree buffers* which install their own major mode named `tree-mode` (the `tree` prefix is one of the prefixes used in the browser Lisp library to avoid name conflicts with other packages.)

Most Emacs keys work in tree buffer mode in the usual way, e.g., you can move around in the buffer with the usual `C-f`, `C-v` etc., or you can search incrementally with `C-s`.

Since tree buffers are read-only most tree buffer specific commands are bound to simple keystrokes, similar to `dired` mode. You can take a look at the key assigments by entering `?` which calls `M-x describe-mode` in both tree and member buffers.

7.1 Viewing and Finding Class Declarations

You can view or find a class declaration with a single keystroke in the line containing the class name.

`v` This command views the declaration of the class in another buffer if the database contains informations about it. If you don't parse the entire source you are working on, some classes will only be known to exist but the location of their definitions will not be known.

`f` Works like `v`, except that it finds the class declaration in another buffer rather than viewing it, so that it is ready for editing.

`SPC` Is the same as `v`.

`RET` Is the same as `f`.

The same functionality is available from the *object menu* opened with `mouse-2` on the class name.

7.2 Displaying Members

There are six different sorts of members, each of which is displayed as a separate *member list*: member instance variables, member instance functions, static (class) variables, static member functions, friend functions, and locally defined (nested) types.

Each of these lists can be displayed in a member buffer with a single keystroke in the line containing the class name. By default, there is only one member buffer named **Members** that is reused each time you display a member list—this has proven to be more practical than to clutter up the buffer list with dozens of member buffers.

If you want to display more than one member list at a time you can *freeze* its member buffer. Freezing a member buffer prevents it from being overwritten the next time you display a member list. You can toggle this buffer status at any time.

Every member list display command in the tree buffer can be used with a prefix argument (`C-u`). Without a prefix argument, the command will pop to a member buffer displaying the member list. With prefix argument, the member buffer will additionally be *frozen*.

`a` This command displays the list of instance member variables.

`A` Display the list of static member variables.

F Display the lits of friend functions.

p Display the list of instance member functions (the '`p`' comes from '`procedures`').

P Display the list of static member functions.

t Display a list of types (`enum`s and `typedef`s) defined with class scope.

mouse-2 Under Emacs 19 a double click with *mouse-2* on a class is a synonym for *p*.

These lists are also available from the class' object menu invoked with *mouse-2* on the class name.

7.3 Finding a Class

g This command reads a class name from the minibuffer with completion and positions the cursor on the class in the class tree.

If the branch of the class tree containing the class searched for is currently collapsed, the class itself and all its base classes are recursively made visible. (See also See Section 7.6 [Expanding and Collapsing], page 21.)

This function is also available from the tree buffer's object menu.

n Repeat the last search done with *g*. Every tree buffer has its own local copy of the regular expression last searched in it. This command can be used to find all occurrences of a class with multiple base classes in the tree buffer.

7.4 Killing or Burying a Tree Buffer

q Is a synonym for *M-x bury-buffer*.

7.5 Displaying File Names

S This command toggles the display of file names in the tree buffer. If file name display is switched on, the names of the files containing the class declaration are shown to the right of the class names. If the file is not known, the string '`unknown`' is displayed.

This command is also provided in the tree buffer's object menu.

s Display file names for the current line, or for the number of lines given by a prefix argument. When Emacs 19 is in use under a windowing system and hilighting is turned on the speed of the display operation is dominated by the highlighting speed which can be rather slow. That is the reason to include a command that avoids excessive redisplay.

Here is an exmaple of a tree buffer with file names displayed.

```
|  Collection <unknown>
|    IndexedCollection <indexedcltn.h>
|      Array <array.h>
|        FixedArray <fixedarray.h>
|    Set <set.h>
|    Dictionary <dict.h>
```

7.6 Expanding and Collapsing a Tree

You can expand and collapse parts of the class tree to reduce the complexity of large class trees. Expanding or collapsing branches of a tree has no impact on the functionality of other commands, like *g*. (see also See Section 7.3 [Go to Class], page 20.)

Collapsed branches are displayed with an ellipsis behind the class name like in the example below.

```
|  Collection
|    IndexedCollection...
|    Set
|    Dictionary
```

`-` This command collapses the branch of the tree starting at the class the cursor is on.

`+` This command expands the branch of the tree starting at the class the cursor is on. Both commands for collapsing and expanding branches are also available from the class' object menu.

`*` This command expands all collapsed branches in the tree. With a prefix argument (*C-u **) all branches are collapsed. You can do the same thing via the buffer's object menu.

mouse-1 If you are running Emacs 19 under a windowing system, a double click with `mouse-1` (usually the left mouse button) will expand or collapse the branch you click on.

7.7 Changing the Tree Indentation

w This command reads a new indentation width from the minibuffer and redisplays the tree buffer with the new indentation. It is also available from the class' object menu.

7.8 Removing Classes from the Tree

\C-d This command removes the class the cursor is on and all its derived classes from the tree. The user is asked for confirmation before the deletion is actually done. After the deletion the buffer is marked as modified.

7.9 Saving a Tree

\C-x\C-s This command writes the class tree in the tree buffer to the file it was read from. This is usefull after classes and/or members have been deleted from or added to the tree.

\C-x\C-w Writes the tree to a file whose name is read from the minibuffer.

x Display statistics for the tree, like number of classes in it, number of member functions, etc. This command can also be found in the buffer object menu.

Classes can be marked for operations similar to the standard Emacs commands *M-x tags-search* and *M-x tags-query-replace* (see also See Chapter 9 [Tags-like Functions], page 27.)

m Toggle the mark of the line point is in or for as many lines as given by a prefix command. This command can also be found in the class' object menu.

u With prefix argument *\C-u*, mark all classes in the tree. Otherwise, unmark all classes. Since this command operates on the whole buffer it can also be found in the buffer's object menu.

Marked classes are displayed with an `>` in column one of the tree display like in the following example

```
|> Collection
|    IndexedCollection...
|>   Set
|    Dictionary
```

8 Member Buffers

Member buffers are used to operate on lists of members of a class. The browser knows six kinds of lists:

- Instance variables (normal member variables),
- Instance functions (normal member functions),
- Static variables,
- Static member functions,
- Friend functions,
- Types (`enum`s and `typedef`s defined with class scope. Nested classes will be shown in the class tree like normal classes.

Like tree buffers, member buffers install their own major mode `member-mode`, whose commands are explained below. Also like in tree buffer, object menus are provided for the different object types in the buffer: members, classes, and the buffer itself.

8.1 Switching Member Lists

`+` This command switches the member buffer to display the next member list.

`-` This command switches the member buffer to display the previous member list.

Both commands cycle through the member list.

Equivalent commands are available from the object menu of the buffer itself. Invoke this menu by clicking *`mouse-2`* in a region of the buffer that is not highlighted (not on a member or class name).

8.2 Finding and VIewing Member Source

`f` This command finds the definition of the member the cursor is on. Finding involves roughly the same as the standard Emacs tags facility does: Loading the file and searching for a regular expression matching the member.

`F` This command finds the declaration of the member the cursor is on.

`v` This is the same command as *`f`*, but views the member definition instead of finding the file the definition is in.

`V` This is the same command as *`F`*, but views the member declaration instead of finding the file the declaration is in.

`SPC` A synonym for *`v`*.

`RET` A synonym for *`f`*.

You can install a hook function to perform actions after a member or class declaration or definition has been found, or when it is not found (see also See Chapter 13 [Customization], page 37.

All these commands can also be found in the object menu displayed when clicking *`mouse-2`* on a member name.

8.3 Display of Inherited Members

`*` This command toggles the display of inherited members in the member buffer. This is also in the buffer's context menu.

8.4 Searching Members

`M-g` Position the cursor on a member whose name is read from the minibuffer; only members shown in the current member buffer appear in the completion list.

`g` Like the above command, but all members for the current class appear in the completion list. If necessary, the current member list is switched to the one containing the member.

With a prefix argument (`\C-u`), all members in the class tree, i.e., all members the browser knows about appear in the completion list. The member display will be switched to the class and member list containing the member.

Look into the buffer's object menu for a convenient way to do this with a mouse.

8.5 Switching to Tree Buffer

`TAB` Pop up the tree buffer to which the member buffer belongs.

`t` Do the same as `TAB` but also position the cursor on the class displayed in the member buffer.

8.6 Filters

`1` This command toggles the display of `public` members.

`2` This command toggles the display of `protected` members.

`3` This command toggles the display of `private` members.

`4` This command toggles the display of `virtual` members.

`5` This command toggles the display of `inline` members.

`6` This command toggles the display of `const` members.

`7` This command toggles the display of `pure virtual` members.

`0` This command removes all filters.

These commands are also in the buffer's object menu.

8.7 Displaying Member Attributes

`a` Show additional information about members:

- Is the member `virtual`? If yes, a `v` is displayed. If no, a `-` is displayed.
- Is the member `inline`? If yes, an `i` is displayed. If no, a `-` is displayed.
- Is the member a `const` function? If yes, a `c` is displayed. If no, a `-` is displayed.

- Is the member a `pure virtual` function? If yes, an `0` is displayed. If no, a `-` is displayed.

These commands are also in the buffer's object menu.

8.8 Long and Short Member Display

L This command switches the member buffer back and forth between short and long display form. The short display form displays member names, only:

```
| isEmpty        contains       hasMember      create
| storeSize      hash           isEqual        restoreGuts
| saveGuts
```

The long display shows one member per line with member name and regular expressions matching the member (if known):

```
| isEmpty               Bool isEmpty () const...
| hash                  unsigned hash () const...
| isEqual               int isEqual (...
```

Regular expressions will only be displayed when the Lisp database has been produced with the `ebrowse` option '`-x`'.

8.9 Display of Regular Expressions

r This command toggles the long display form from displaying the regular expressions matching the member declarations to those expressions matching member definitions.

Regular expressions will only be displayed when the Lisp database has been produced with the `ebrowse` option '`-x`'.

8.10 Displaying Another Class

c This command lets you switch the member buffer to another class. It reads the name of the new class from the minibuffer with completion.

u This is the same command as *c* but restricts the classes shown in the completion list to immediate base classes, only. If only one base class exists, this one is immediately shown in the minibuffer.

d Same as *u*, but for derived classes.

p Switch to the previous class in the class hierarchy on the same level as the class currently displayed.

n Switch to the next sibling of the class in the class tree.

8.11 Killing and Burying a Member Buffer

q This command is a synonym for *M-x bury-buffer*.

8.12 Setting the Column Width

`w` This command sets the column width depending on the display form used (long or short display).

8.13 Forced Redisplay

`l` This command forces a redisplay of the member buffer. If the width of the window displaying the member buffer is changed this command redraws the member list with the appropriate column widths and number of columns.

`?` This key is bound to `describe-mode` which display the current key bindings in another window.

9 Tags-like Functions

The *C++ class browser* provides tags functions similar to those of the standard Emacs distribution file `tags.el`, but better suited to the needs of C++ programmers.

In the following descriptions two keys are listed for each available command. Keys prefixed with `\C-c b` are only defined if you `(require 'br-utils)` in your `.emacs`. (See also See Section 4.5 [Startup], page 12).

9.1 Finding and Viewing Members

M-x browse-tags-find, \C-c b .
: When standing on an identifier in some source file, you can call this function to find the declaration or definition of the identifier. If you invoke this function with a prefix argument (*C-u*), the declaration is searched.

  You can use completion for the identifier, if the variable `browse-fast-member-lookup` is non-nil which is the default. (See also See Chapter 13 [Customization], page 37.)

  If more than one class contains an identifier/ member with the given name you can select the class with completion. If there is a scope declaration in front of the member name this class name is read and used as initial input for the completion.

M-x browse-tags-view, \C-c b v
: This command works like *M-x browse-tags-find*, except that is views the member's declaration or definition.

9.2 The Position Stack

When jumping to a member declaration or definition with *M-x browse-tags-find* or *M-x browse-tags-view*, the position from where you performed the jump and the position where you jumped to are recorded in a *position stack*. There are several ways in which you can quickly move to positions in the stack:

M-x browse-tags-back, \C-c b b
: This command sets point to the previous position in the position stack. Directly after you performed a jump, this will put you back to the position where you came from. If the buffer the recorded position is in is currently not in memory, it is loaded.

  The stack is not popped, i.e., you can always switch back and forth between positions in the stack. To avoid letting the stack grow to infinite size there is a maximum number of positions defined. When this number is reached, older positions are discarded when new positions are pushed on the stack (see also See Chapter 13 [Customization], page 37.)

M-x browse-tags-forward \C-c b f
: This command moves forward in the position stack, setting point to the next position stored in the position stack.

M-x browse-tags-position-list, \C-c b p
: Displays an electric buffer showing all positions saved in the stack. You can select a position by pressing *SPC* in a line. You can view a position with *v*.

9.3 Searching and Replacing

The *C++ browser* allows you to perform operations on all or a subset of the files containing class and member declarations or definitions of a class tree. When you invoke one of the following functions and more than one class tree is loaded, you must choose a class tree to use from an electric tree menu. If the selected tree contains marked classes, the following commands operate on the files mentioned in the marked classes only. Otherwise all files in the class tree are used.

M-x browse-search, \C-c b s
: This function performs a regular expression search in the chosen set of files.

M-x browse-search-member-usage, \C-c b u
: This command performs a search for calls of a given member that is selected in the usual way with completion.

M-x browse-query-replace, \C-c b %
: Perform a query replace over the set of files.

M-x browse-loop, \C-c b ,
: All three operations above stop when finding a match. You can restart the operation with this command.

M-x browse-next-file, \C-c b n
: This restarts the last tags operation with the next file in the list.

9.4 Members in Files

The command *M-x browse-tags-list*, *\C-c b l*, displays all members in a given file whose name is read from the minibuffer with completion.

9.5 Member Usage

The command *M-x browse-search-member-usage* is a special form of *M-x browse-search* that searches for a regular expression that looks like a call to a given member. This command is bound to *\C-c b u* by `br-utils`.

9.6 Member Apropos

The command *browse-tags-apropos*, *\C-c b a* can be used to display all members matching a given regular expression. This command can be very usefull if you remember only a part of a member name, and not its beginning.

A special buffer is popped up containing all identifiers matching the regular expression, and what kind of symbol it is (e.g., a member function, or a type). You can switch to this buffer, and use the command *M-x browse-tags-find* to jump to a specific member.

9.7 Symbol Completion

The command *M-x browse-complete-symbol*, which is bound to *C-tab* under Emacs 19 in C++ buffers by `br-utils.el` completes the symbol in front of point.

9.8 Quick Member Display

You can quickly display a member buffer containing the member the cursor in on with the command *M-x browse-tags-find-member-buffer*; this command is on *\C-c b m* if you use use `br-utils.el`.

10 Utilities

The file `br-utils.el` contains several utilities like key bindings that are usefull when using the C++ browser.

11 Global Key Bindings

All global keys are prefixed with `\C-c b`. Here is a list of what is defined when you load `br-utils`:

`a`	*M-x browse-tags-apropos*
`b`	*M-x browse-tags-back*
`f`	*M-x browse-tags-forward*
`l`	*M-x browse-tags-list*
`m`	*M-x browse-tags-find-member-buffer*
`n`	*M-x browse-next-file*
`p`	*M-x browse-electric-position-list*
`s`	*M-x browse-search*
`u`	*M-x browse-search-member-usage*
`v`	*M-x browse-tags-view*
`%`	*M-x browse-query-replace*
`.`	*M-x browse-tags-find*
`,`	*M-x browse-loop*
`+ r`	*M-x browse-add-region*
`+ b`	*M-x browse-add-buffer*

12 Electric Tree Buffers

The command `\C-c b SPC` pops up an electric buffer list containing browser related buffers only.

13 Customization

13.1 Verbose Mode

On slow machines when working with large class hierarchies, it is sometimes a good idea to let the browser display what it is doing while loading a database into memory. This can be switched off by something like

```
(setq browse-options (remq 'verbose browse-options))
```

somewhere in your `.emacs` after the browser has been loaded.

13.2 Fast Member Lookup

The variable `browse-fast-member-lookup` makes the browser use a faster member lookup scheme that cost additional memory. If you are short of RAM or working on very large class hierarchies, you may want to turn this option off by setting the variable to NIL.

13.3 Highlighting

The browser normally highlightes tree and member buffers when Emacs 19 is used under a windowing system (it doesn't use the `hilight.el` package to do this because it can be done more efficient in these special cases). You can turn the highlighting off by setting the variable `browse-hilit-on-redisplay` to `nil`.

13.4 Colors

The colors used for highlighting tree and member buffers can be changed by setting the following variables:

`tree-mark-face`
: This is the face of the mark sign `>` in tree buffers.

`tree-root-class-face`
: The face used for the display of root classes.

`tree-multiply-derived-face`
: The face used for classes with more than one base class.

`tree-filename-face`
: The face for the filename display in tree buffers.

`tree-normal-face`
: The face for normal class names.

`member-attributes-faces`
: The face used for attributes display in member buffers.

`member-class-face`
: The face used for class titles in member buffers.

You can set these faces by installing an appropriate `tree-mode-hook` or `member-mode-hook`

13.5 Hooks

The following hooks can be installed:

`browse-find-hook`
: Run after finding or viewing a definition or declaration.

`browse-not-found-hook`
: Run when finding or viewing was not successfull.

`member-mode-hook`
: Run when switching a buffer to `member-mode`.

`browse-electric-position-mode-hook`
: Run for electric position buffers.

`tree-mode-hook`
: Hook for tree buffers.

Here is an expample of using `browse-find-hook` to do something that might be usefull to some degree. when finding or viewing a member. it narrows the buffer containing the function to just the function definition.

```
(defun test-hook ()
  (if (save-excursion (beginning-of-line)
      (and (looking-at "^.*(")
   (not (looking-at "^.*;"))))
      (progn
;; function definition assumed
(let ((org (point)) start)
  ;; Skip backward until we reach a line that doesn't start with
  ;; a char that might be part of the name or type.
  (while (progn (forward-line -1)
(looking-at "^[a-zA-Z_]")))
  (setq start (point))
  ;; Find opening brace of function definition.
  (when (search-forward "{" nil t)
    (goto-char (1- (point)))
    (forward-list)
    (narrow-to-region start (point))
    (goto-char (min start (max (point) org)))
    (message "Use %s to see the whole buffer..."
     (key-description (where-is-internal 'widen nil t)))
    (sit-for 2))
  (goto-char org)))
    (recenter)))

(add-hook 'browse-find-hook 'test-hook)
```

14 Concept Index

Concept Index

O

P

R

S

T

U

V

Table of Contents