Ebrowse User's Manual

Ebrowse/Emacs 21

May 2000

Gerd Moellmann

1 Introduction

When working in software projects using C++, I frequently missed software support for two things:

- When you get a new class library, or you have to work on source code you haven't written yourself (or written sufficiently long ago), you need a tool to let you navigate 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 Ebrowse reflects these two needs.

How does it work?

A fast parser written in C is used to process C++ source files. The parser generates a data base containing information about classes, members, global functions, defines, types etc. found in the sources.

The second part of Ebrowse is a Lisp program. This program reads the data base generated by the parser. It displays its contents in various forms and allows you to perform operations on it, or do something with the help of the knowledge contained in the data base.

Navigational use of Ebrowse is centered around two types of buffers 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 class declarations, perform operations like query replace on sets of your source files, and finally tree buffers are used to produce the second buffer form—member buffers. See Chapter 4 [Tree Buffers], page 9.

Members are displayed in *member buffers*. Ebrowse 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/Defines. The list of defines is contained in the friends list of the pseudo-class '`*Globals*`';
- Types (`enum`s, and `typedef`s defined with class scope).

You can switch member buffers from one list to another, or to another class. You can include inherited members in the display, you can set filters that remove categories of members from the display, and most importantly you can find or view member declarations and definitions with a keystroke. See Chapter 5 [Member Buffers], page 13.

These two buffer types and the commands they provide support the navigational use of the browser. The second form resembles Emacs' Tags package for C and other procedural languages. Ebrowse's commands of this type are not confined to special buffers; they are most often used while you are editing your source code.

To list just a subset of what you can use the Tags part of Ebrowse for:

- Jump to the definition or declaration of an identifier in your source code, with an electric position stack that lets you easily navigate back and forth.
- Complete identifiers in your source with a completion list containing identifiers from your source code only.
- 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.

2 Processing Source Files

Before you can start browsing a class hierarchy, you must run the parser `ebrowse` on your source files in order to generate a Lisp data base describing your program.

The operation of `ebrowse` can be tailored with command line options. 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 shell doesn't allow all the file names to be specified on the command line,

```
ebrowse --files=file
```

where *file* contains the names of the files to be parsed, one per line.

When invoked with option '`--help`', `ebrowse` prints a list of available command line options.

2.1 Specifying Input Files

'`file`' Each file name on the command line tells `ebrowse` to parse that file.

'`--files=`*file*'
: This command line switch specifies that *file* contains a list of file names to parse. Each line in *file* must contain one file name. More than one option of this kind is allowed. You might, for instance, want to use one file for header files, and another for source files.

'`standard input`'
: When `ebrowse` finds no file names on the command line, and no '`--file`' option is specified, it reads file names from standard input. This is sometimes convenient when `ebrowse` is used as part of a command pipe.

'`--search-path=`*paths*'
: This option lets you specify search paths for your input files. *paths* is a list of directory names, separated from each other by a either a colon or a semicolon, depending on the operating system.

It is generally a good idea to specify input files so that header files are parsed before source files. This facilitates the parser's work of properly identifying friend functions of a class.

2.2 Changing the Output File Name

'`--output-file=`*file*'
: This option instructs `ebrowse` to generate a Lisp data base with name *file*. By default, the data base is named '`BROWSE`', and is written in the directory in which `ebrowse` is invoked.

: If you regularly use data base names different from the default, you might want to add this to your init file:

```
(add-to-list 'auto-mode-alist '(NAME . ebrowse-tree-mode))
```

where *NAME* is the Lisp data base name you are using.

‘`--append`’
: By default, each run of `ebrowse` erases the old contents of the output file when writing to it. You can instruct `ebrowse` to append its output to an existing file produced by `ebrowse` with this command line option.

2.3 Structs and Unions

‘`--no-structs-or-unions`’
: This switch suppresses all classes in the data base declared as `struct` or `union` in the output.

  This is mainly useful when you are converting an existing C program to C++, and do not want to see the old C structs in a class tree.

2.4 Regular Expressions

The parser `ebrowse` normally writes regular expressions to its output file that help the Lisp part of Ebrowse to find functions, variables etc. in their source files.

You can instruct `ebrowse` to omit these regular expressions by calling it with the command line switch ‘`--no-regexps`’.

When you do this, the Lisp part of Ebrowse tries to guess, from member or class names, suitable regular expressions to locate that class or member in source files. This works fine in most cases, but the automatic generation of regular expressions can be too weak if unusual coding styles are used.

‘`--no-regexps`’
: This option turns off regular expression recording.

‘`--min-regexp-length=`*n*’
: The number *n* following this option specifies the minimum length of the regular expressions recorded 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 Ebrowse 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.

‘`--max-regexp-length=`*n*’
: 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. As a second effect, the larger the regular expression, the higher the probability that it will no longer match after editing the file.

2.5 Verbose Mode

‘`--verbose`’
: When this option is specified on the command line, `ebrowse` prints a period for each file parsed, and it displays a ‘`+`’ for each class written to the output file.

‘`--very-verbose`’
: This option makes `ebrowse` print out the names of the files and the names of the classes seen.

3 Starting to Browse

You start browsing a class hierarchy parsed by `ebrowse` by just finding the ‘`BROWSE`’ file with *`C-x C-f`*.

An example of a tree buffer display is shown below.

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

When you run Emacs on a display which supports colors and the mouse, you will notice that certain areas in the tree buffer are highlighted when you move the mouse over them. This highlight marks mouse-sensitive regions in the buffer. Please notice the help strings in the echo area when the mouse moves over a sensitive region.

A click with *`Mouse-3`* on a mouse-sensitive region opens a context menu. In addition to this, each buffer also has a buffer-specific menu that is opened with a click with *`Mouse-3`* somewhere in the buffer where no highlight is displayed.

4 Tree Buffers

Class trees are displayed in *tree buffers* which install their own major mode. Most Emacs keys work in tree buffers 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 with *`C-s`*.

Tree-specific commands are bound to simple keystrokes, similar to `Gnus`. You can take a look at the key bindings by entering *`?`* which calls `M-x describe-mode` in both tree and member buffers.

4.1 Viewing and Finding Class Declarations

You can view or find a class declaration when the cursor is on a class name.

`SPC` This command views the class declaration 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 declarations and definitions will not be known.

`RET` Works like *`SPC`*, except that it finds the class declaration rather than viewing it, so that it is ready for editing.

The same functionality is available from the menu opened with *`Mouse-3`* on the class name.

4.2 Displaying Members

Ebrowse distinguishes six different kinds of members, each of which is displayed as a separate *member list*: instance variables, instance functions, static variables, static functions, friend functions, and types.

Each of these lists can be displayed in a member buffer with a command starting with *`L`* when the cursor is on a 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*.

`L v` This command displays the list of instance member variables.

`L V` Display the list of static variables.

`L d` Display the list of friend functions. This list is used for defines if you are viewing the class '`*Globals*`' which is a place holder for global symbols.

`L f` Display the list of member functions.

`L F` Display the list of static member functions.

L t Display a list of types.

These lists are also available from the class' context menu invoked with *Mouse-3* on the class name.

4.3 Finding a Class

/ 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 Section 4.6 [Expanding and Collapsing], page 10.)

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

n Repeat the last search done with */*. Each tree buffer has its own local copy of the regular expression last searched in it.

4.4 Burying a Tree Buffer

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

4.5 Displaying File Names

T f This command toggles the display of file names in a 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 context menu.

s Display file names for the current line, or for the number of lines given by a prefix argument.

Here is an example 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)
```

4.6 Expanding and Collapsing a Tree

You can expand and collapse parts of a tree to reduce the complexity of large class hierarchies. Expanding or collapsing branches of a tree has no impact on the functionality of other commands, like */*. (See also Section 4.3 [Go to Class], page 10.)

Collapsed branches are indicated with an ellipsis following 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.

4.7 Changing the Tree Indentation

T 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 tree buffer's context menu.

4.8 Removing Classes from the Tree

C-k 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 performed.

4.9 Saving a Tree

C-x C-s This command writes a class tree to the file from which it was read. This is useful after classes have been deleted from a 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's context 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 6 [Tags-like Functions], page 17.)

M t 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' context menu.

M a Unmark all classes. With prefix argument *C-u*, mark all classes in the tree. 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
```

5 Member Buffers

Member buffers are used to operate on lists of members of a class. Ebrowse distinguishes 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. Also like in tree buffers, menus are provided for certain areas in the buffer: members, classes, and the buffer itself.

5.1 Switching Member Lists

`L n`	This command switches the member buffer display to the next member list.
`L p`	This command switches the member buffer display to the previous member list.
`L f`	Switch to the list of member functions.
`L F`	Switch to the list of static member functions.
`L v`	Switch to the list of member variables.
`L V`	Switch to the list of static member variables.
`L d`	Switch to the list of friends or defines.
`L t`	Switch to the list of types.

Both commands cycle through the member list.

Most of the commands are also available from the member buffer's context menu.

5.2 Finding and Viewing Member Source

`RET`	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.
`SPC`	This is the same command as *`RET`*, but views the member definition instead of finding the member's source file.
`v`	This is the same command as *`f`*, but views the member's declaration instead of finding the file the declaration is in.

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.

All the commands described above can also be found in the context menu displayed when clicking *`Mouse-2`* on a member name.

5.3 Display of Inherited Members

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

5.4 Searching Members

G v	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 m	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.
G n	Repeat the last member search.

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

5.5 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.

5.6 Filters

F `a` *u*	This command toggles the display of `public` members. The '`a`' stands for 'access'.
F `a` *o*	This command toggles the display of `protected` members.
F `a` *i*	This command toggles the display of `private` members.
F v	This command toggles the display of `virtual` members.
F i	This command toggles the display of `inline` members.
F c	This command toggles the display of `const` members.
F p	This command toggles the display of pure virtual members.
F r	This command removes all filters.

These commands are also found in the buffer's context menu.

5.7 Displaying Member Attributes

D a Toggle the display of member attributes (default is on).

The nine member attributes Ebrowse knows about are displayed as a list a single-characters flags enclosed in angle brackets in front the of the member's name. A '`-`' at a given position means that the attribute is false. The list of attributes from left to right is

'`T`'	The member is a template.
'`C`'	The member is declared `extern "C"`.
'`v`'	Means the member is declared `virtual`.
'`i`'	The member is declared `inline`.
'`c`'	The member is `const`.
'`0`'	The member is a pure virtual function.
'`m`'	The member is declared `mutable`.
'`e`'	The member is declared `explicit`.
'`t`'	The member is a function with a throw list.

This command is also in the buffer's context menu.

5.8 Long and Short Member Display

D l This command toggles the member buffer 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 not been produced with the `ebrowse` option '`--no-regexps`'. See Section 2.4 [Regular Expressions], page 4.

5.9 Display of Regular Expressions

D 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 not been produced with the `ebrowse` option '`--no-regexps`', see Section 2.4 [Regular Expressions], page 4.

5.10 Displaying Another Class

C 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.
C b	This is the same command as *C 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.
C d	Same as *C b*, but for derived classes.
C p	Switch to the previous class in the class hierarchy on the same level as the class currently displayed.
C n	Switch to the next sibling of the class in the class tree.

5.11 Burying a Member Buffer

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

5.12 Setting the Column Width

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

5.13 Forced Redisplay

C-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`.

6 Tags-like Functions

Ebrowse provides tags functions similar to those of the standard Emacs Tags facility, but better suited to the needs of C++ programmers.

6.1 Finding and Viewing Members

The functions in this section are similar to those described in Section 4.1 [Source Display], page 9, and also in Section 5.2 [Finding/Viewing], page 13, except that they work in a C++ source buffer, not in member and tree buffers created by Ebrowse.

`C-c b f` Find the definition of the member around point. If you invoke this function with a prefix argument, the declaration is searched.

If more than one class contains a 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 used as initial input for the completion.

`C-c b F` Find the declaration of the member around point.

`C-c b v` View the definition of the member around point.

`C-c b V` View the declaration of the member around point.

`C-c b 4 f` Find a member's definition in another window.

`C-c b 4 F` Find a member's declaration in another window.

`C-c b 4 v` View a member's definition in another window.

`C-c b 4 V` View a member's declaration in another window.

`C-c b 5 f` Find a member's definition in another frame.

`C-c b 5 F` Find a member's declaration in another frame.

`C-c b 5 v` View a member's definition in another frame.

`C-c b 5 V` View a member's declaration in another frame.

6.2 The Position Stack

When jumping to a member declaration or definition with one of Ebrowse's commands, 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:

`C-c 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.

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.

`C-c b +`	This command moves forward in the position stack, setting point to the next position stored in the position stack.
`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`.

6.3 Searching and Replacing

Ebrowse allows you to perform operations on all or a subset of the files mentioned in 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.

`C-c b s`	This function performs a regular expression search in the chosen set of files.
`C-c b u`	This command performs a search for calls of a given member which is selected in the usual way with completion.
`C-c b %`	Perform a query replace over the set of files.
`C-c b ,`	All three operations above stop when finding a match. You can restart the operation with this command.
`C-c b n`	This restarts the last tags operation with the next file in the list.

6.4 Members in Files

The command `C-c b l`, lists all members in a given file. The file name is read from the minibuffer with completion.

6.5 Member Apropos

The command `C-c b a` can be used to display all members matching a given regular expression. This command can be very useful if you remember only 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 then switch to this buffer, and use the command `C-c b f`, for example, to jump to a specific member.

6.6 Symbol Completion

The command `C-c b TAB` completes the symbol in front of point.

6.7 Quick Member Display

You can quickly display a member buffer containing the member the cursor in on with the command `C-c b m`.

Concept Index

Table of Contents