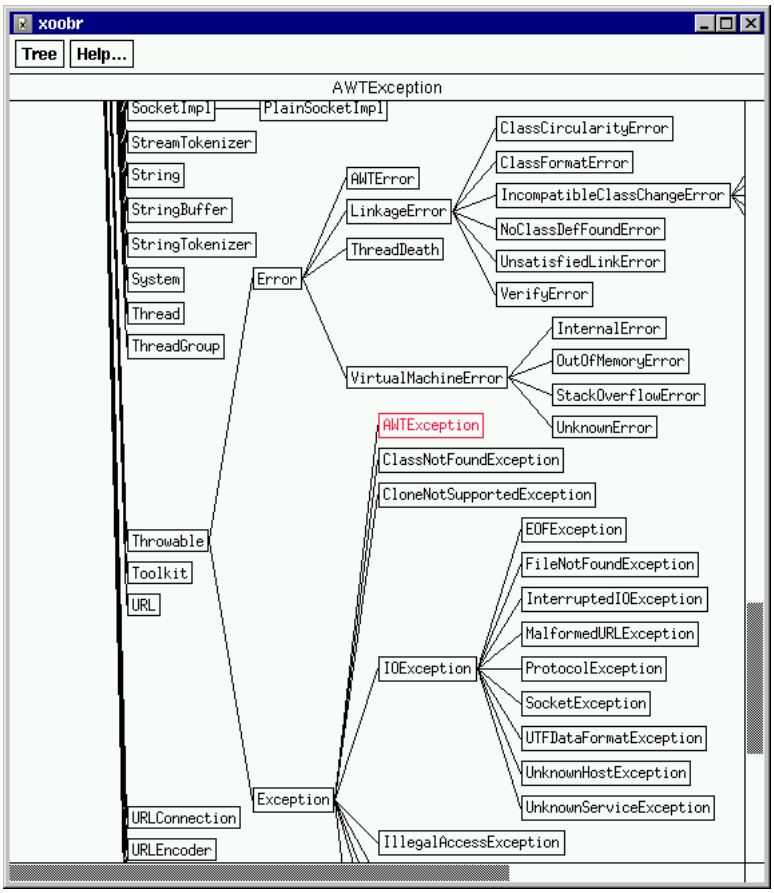
The GNU OO-Browser

The Multi-language Object-Oriented Code Browser



Bob Weiner

This manual is for the GNU OO-Browser (Edition 5.0.2, Published 3 Aug 2016).

Copyright © 1996-2016 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation.

GNU Hyperbole sofware is distributed under the terms of the GNU General Public License version 3 or later, as published by the Free Software Foundation, Inc.

The OO-Browser is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details in the file, "COPYING", within the OO-Browser package directory.

Published by the Free Software Foundation, Inc.
Author: Bob Weiner
E-mail: <oo-browser-users@gnu.org> (This is a mail list.)
Web: http://savannah.nongnu.org/projects/oo-browser

The body of the manual was written in Emacs and laid out using the GNU Texinfo markup language.

Short Contents

Table of Contents

Preface

The OO-Browser was designed and written by Bob Weiner. Motorola, Inc. helped fund early work. Torgeir Veimo and Mark Stern helped write the X OO-Browser core. Jeff Sparkes helped with the Java language support. Harri Pasanen contributed the initial Python language support (derived from the C++ support). Kirill Katsnelson adapted the graphical OO-Browser for use with Windows.

The OO-Browser has been updated to work well with the very latest versions of GNU Emacs.

The latest available release of the OO-Browser is always available for download from `www.sourceforge.net/projects/oo-browser`. They include the OO-Browser with all supported languages and full source code, a printed copy of this manual and installation support. Ongoing technical support and automatic upgrades are available separately.

The OO-Browser must be installed at your site before you can use it. Instructions for installing the OO-Browser are in the `INSTALL` file in the root directory of the OO-Browser distribution, i.e. below `oo-browser/`. If you are using InfoDock, the OO-Browser is pre-installed so you can skip the installation instructions.

We hope you enjoy using the OO-Browser and that it improves your productivity. If it does, consider sending us a quote that discusses how it helps you, for use on our web site. E-mail it to <oo-browser-users@gnu.org>.

Introduction

This edition of the OO-Browser User Manual is for use with any version 4.06 or greater of the OO-Browser.

This manual documents the user interface and operation of the OO-Browser. It assumes a very basic familiarity in the use of InfoDock, XEmacs or Emacs, as documented in *[Stallman 93]*. It also assumes familiarity with object-oriented software concepts. However, many of the technical terms used in this manual are defined within the glossary. See Appendix A [Glossary], page 37.

The OO-Browser is designed to be easy to use. It has point and click and menu-based interfaces that you can use, if you prefer, instead of learning all of the keystroke commands. The body of this manual discusses the mouse and keyboard interfaces. If you would like to study the menus, see Appendix B [Menus], page 41.

Chapter 1 of the manual discusses OO-Browser Environments as a means of organizing browser work (see Chapter 1 [Working with Environments], page 4). See Chapter 2 [Using the OO-Browser], page 8, if you would rather start with the interactive features of the browser. See Appendix C [Features], page 51, for a quick overview of the browser's features.

Throughout this manual, sequences of keystrokes are delimited by curly braces { }, function names are delimited by parentheses ( ), and variable names look like `this`.

The *OO-Browser* (pronounced owe-owe-browse-er) is a multi-windowed, interactive, object-oriented class browser designed for professional use. It is one of the world's most powerful tools for exploring and developing object-oriented software. Its user interface is a bit like the well-known Smalltalk browsers *[Goldberg 83]*, yet its commands are more flexible and easier to use.

The OO-Browser has a number of exceptional features:

- It presently supports seven object-oriented languages (C++, CLOS/Lisp, Eiffel, Java, Objective-C, Python and Smalltalk), one non-object-oriented language (C), and one documentation language (GNU Info).
- It is fast and works exclusively from the source code of libraries and programs to be browsed. Hence, it does not require a compiler.
- It may be used for both system exploration and maintenance as part of a professional software development tool chest.
- It quickly displays views of several important object-oriented relationships, over large sets of classes, not just a single class at a time.
- It has a completely direct-manipulation interface with multiple modalities.
- It is integrated with a powerful editing environment that can be customized to meet personal work styles.

Refer to the picture on the following page as we highlight the major components of the OO-Browser user interface.

The windows across the top of the OO-Browser frame are called *listing windows*; they display *listing buffers* that list class, method and attribute names based on user-specified

queries. The *viewer window* fills the bottom half of the frame. It is used to display class source, summary information and help on the OO-Browser command set. The picture shows part of a Java Boolean class in the viewer window.

All key bindings described throughout this manual are effective only within listing buffers, unless otherwise indicated. This means that the keys may not be used within the buffers displayed in the class viewer window. Instead, all normal editing keys are available in most viewer window buffers.

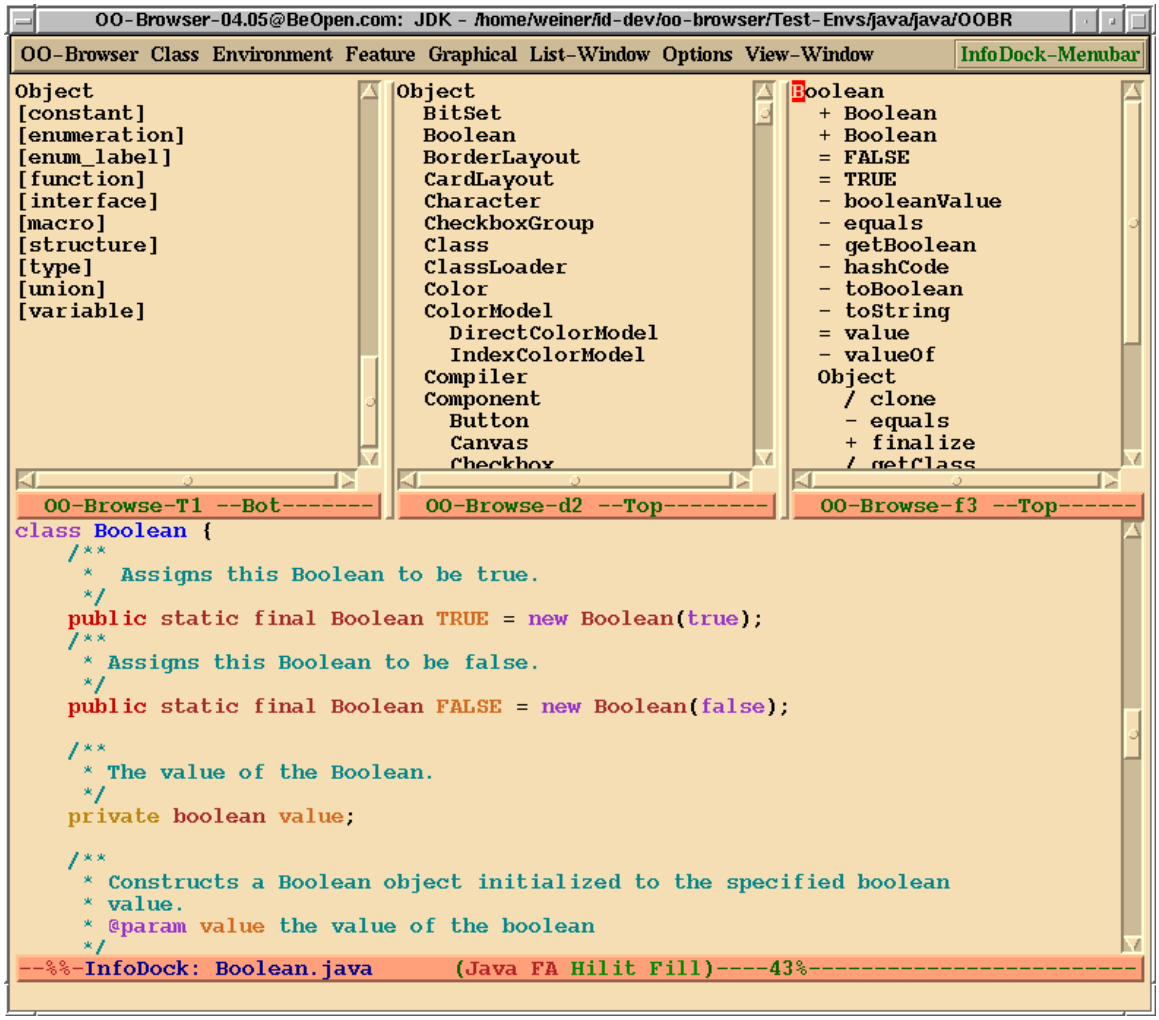


Image 1: The Textual OO-Browser Screenshot

1 Working with Environments

In order to browse code from a set of libraries or an entire system, an OO-Browser Environment must first be created. An OO-Browser *Environment* defines the set of entities that are browsable within a session of the browser. The first step in creating an Environment is to create an *Environment specification* which tells the browser the name of the Environment, where to save its browsing data, what programming language the Environment should support, and what directories to scan for source code to browse. (See Section 1.1 [Creating Environments], page 4, for more information.)

Once an Environment has been specified, it can be built, a process which scans the set of directory trees given in the Environment specification and saves data about classes, methods, attributes, formal protocols (interfaces) and relationships among these entities for later browsing.

Whenever the OO-Browser is in use, an Environment is selected. The phrase, *the Environment*, refers to the current OO-Browser Environment. Many browser commands depend upon information in the Environment.

The set of source files included in an Environment is specified by two lists of top-level (root) directories whose subdirectories are automatically scanned by the OO-Browser. The first list of directories is called the *system-specific directory list*; it defines the locations of unreleased code being developed, often for a particular system. The second list is called the *reusable library directory list*; it defines the locations of stable, reusable classes that have been released for general use. All class names within a single Environment must be unique to ensure proper operation of the browser (Duplicate classes will be flagged by the browser for later removal. Use {`M-e`} `(br-env-stats)` in a browser listing window to help find duplicate classes.)

The OO-Browser lets you name, create, update and save Environments. Once an Environment has been created, it may be loaded at any time. The browser will then use this Environment for all of its operations until another one is loaded.

Within each editor session, the browser caches a separate Environment for each programming language on which it is used. Thus, if you switch from Eiffel to C++ browsing and then back to Eiffel browsing, the Eiffel environment will not need to be reloaded; it will appear immediately and the frame will appear as if the Eiffel OO-Browser were invoked for the first time.

Environment files are automatically saved by the OO-Browser so that you need never become familiar with their format. You are responsible for requesting the use of a particular Environment whenever desired. See Section 2.1 [Invoking the OO-Browser], page 8, for information on how to specify a different Environment file for use.

1.1 Creating Environments

Environments may be specified, built and/or loaded at any time all with one command, whether or not the browser user interface is on screen. Use the {`C-c o`} `(oo-browser)` command to select a language and Environment to create or to load by name. (Some programming modes may override that key binding, so a menu item or '`M-x oo-browser RET`' are the surefire ways to invoke this command.)

If you have browsed a prior Environment during your current editor session, you will be asked whether or not to reload that Environment. If you type '`n`' for no or if no prior Environment has been loaded, you will be prompted for an Environment name to load; full completion is available.

If instead of typing an Environment name, you simply press Return, you will be prompted for the pathname of an Environment file to load. This lets you specify Environments created prior to version 4 of the OO-Browser, which did not have names. The default is to load the Environment file in the current directory whose name is given by the `br-env-default-file` variable, which is normally set to `OOBR`.

If you give an Environment name or file which does not exist, you will be prompted to create a specification for this new Environment. The recommended default name for Environment files is, `OOBR`. It is best to save each Environment within the top-level directory of the first system-specific directory of the Environment, i.e. the root directory of a system's code.

Every Environment must be specified before it can be built or used. Here are the Environment specification components for which you will be prompted:

`Top-level system-specific code directories`
: A list of root directories below which system source code may be found. Each directory is prompted for individually and argument completion is available.

`Top-level reusable library code directories`
: A list of root directories below which library source code may be found. Each directory is prompted for individually and argument completion is available.

Environment specifications are useful when you want to describe a number of Environments to the OO-Browser yet also want to defer their construction until later. Large environments then can be built overnight. When in an OO-Browser listing window, multiple Environment specifications may be created in quick succession with the {`C-c C-c`} `(br-env-browse)` command, which prompts for all necessary information to create an Environment specification; simply give it a non-existing Environment name and answer no when prompted to build the Environment to defer building for later. The OO-Browser will automatically prompt you to build an Environment specification whenever you try to load it for browsing.

1.2 Building Environments

An Environment specification tells the OO-Browser what to include in the Environment, but the Environment must be built by scanning the source directories before it can be used. When a new Environment is to be built or when a large number of changes have been made to classes in the current Environment, the following commands are useful:

{`C-c C-e`} `(br-env-rebuild)` within an OO-Browser
: listing window scans all Environment source code directories. This prompts for whether or not to use a background process to build the Environment (if a '`make`' program is available), so that other tasks may be performed while the build is active. With a prefix argument under InfoDock or XEmacs, {`C-u C-c C-e`}, it runs the build in the background and produces a stack backtrace if an error occurs during the build;

{L} (br-lib-rebuild)
: scans only the reusable library directories of the Environment;

{S} (br-sys-rebuild)
: scans only the system-specific directories of the Environment.

Whenever class inheritance relations or the set of attributes or methods for a class changes, the Environment must be rebuilt, but this is generally a fast process.

The OO-Browser lets you build large Environments in the background of your interactive session (if a 'make' program is available), allowing other work to proceed without waiting on a build. When a build is complete, the OO-Browser will prompt you and give you a choice of whether or not to browse the built Environment immediately.

Alternatively, any number of very large Environments may be built overnight by invoking Emacs in batch mode from a shell script. To do this, you must first create an Environment specification so that the browser knows what to build. See Section 1.1 [Creating Environments], page 4. Then use a shell command line of the form below (substitute your local OO-Browser installation directory for *Br-Dir*):

```
cd Br-Dir; emacs -batch -l ./oo-browser.elc Env-Spec-File-1 \
   ... Env-Spec-File-N -f br-env-batch-build > Log-File
```

for example:

```
cd oo-browser; emacs -batch -l ./oo-browser.elc ~/OOBR \
   -f br-env-batch-build > log-file
```

Redirection of the standard output stream to a log file for later examination helps ensure that either the Environment build is successful or an error is logged.

1.3 Saving Environments

The OO-Browser automatically builds and saves Environments in most cases. Occasionally you may find a need to force the Environment to be saved to a file, as in the case when you want to save an Environment under a different file name.

Use {C-c C-s}, the (br-env-save) command, to force a save of the current Environment. The command will prompt for a file to save to, with the present Environment file name as the default.

1.4 Managing Environment Names

The OO-Browser offers a set of menu and keyboard-based commands for managing user-specific names associated with Environment files. The menu commands are found on the Environment menu within browser listing buffers. See Appendix B [Menus], page 41. The related keyboard commands are for use within listing buffers.

Each time you create an Environment, you give it a memorable name which is stored with the associated Environment file name, in a file given by the variable, br-names-file. This variable is set to ~/.oo-browser on UNIX-like OSes and c:/_oo-browser on Microsoft operating systems. This *Environment names file* is loaded by the OO-Browser

at startup so that it can offer completion to assist you when entering Environment names. The Environment names file is automatically saved whenever necessary by the browser.

Once an Environment has been loaded, your full list of Environment names can be displayed with the {M-l} (br-names-display) command. This shows each name together with its Environment file name.

If you want to browse an Environment created by another user or want to add additional names to an Environment, when in the browser, use the {M-a} (br-name-add) command. This will prompt for a name and an existing Environment file with which to associate the name.

Use the {M-n} (br-name-change) command to rename an Environment. The command, {M-m} (br-name-remove), will delete an existing name. It removes the name only, not the Environment itself. The {M-r} (br-name-replace) command will change the Environment file associated with a particular name.

2 Using the OO-Browser

2.1 Invoking the OO-Browser

The OO-Browser supports the following languages: C++, C, CLOS (Lisp), Eiffel, Java, Info (the online manual format), Objective-C, Python and Smalltalk. Use {`C-c o`} or, if that key has not been setup, use '`M-x oo-browser RET`' to browse source written in any of the above languages. This command will prompt for the name of an existing or new Environment to browse, and then will either create, build or load the Environment, depending on the state of the Environment specification. After the Environment is built, the browser will display the entire set of classes defined or referenced within the Environment. (Choose C++ as the language if you are browsing plain C code.)

Alternatively, you can invoke the browser on a specific language Environment, e.g. to bring back the last Environment browsed under that language. The language-specific browser invocation commands are: '`M-x c++-browse RET`', '`M-x clos-browse RET`', '`M-x eif-browse RET`', '`M-x info-browse RET`', '`M-x java-browse RET`', '`M-x objc-browse RET`', '`M-x python-browse RET`', and '`M-x smt-browse RET`'. A prefix argument given to any of these commands will cause it to prompt for an Environment file to use, rather than automatically reusing the last Environment of the same language.

On startup, if the named Environment exists and is built, it will be loaded; otherwise, you will be asked to specify the Environment. The specification will be saved under the previously given file name.

If the browser loads an Environment file and finds only a specification, it will prompt you in the minibuffer window with a request to build the Environment. It will continue to prompt you until a full Environment is built or loaded. It then will start interactive operation, displaying its multi-windowed interface. To abort from these prompts and to cancel the browser invocation request at any time, use {`C-g`} `(keyboard-quit)`, the standard way to abort an unfinished command within Emacs and InfoDock.

The first time you start the OO-Browser during each editor session, it will display a version and credits screen within the viewer window. When you press any key, the screen will be replaced by the keyboard command help screen of the OO-Browser and the command associated with the key you pressed will be executed. If you'd like to read the rest of the credits, you may redisplay the version screen at any time from within a browser listing window by using {`C-c #`} `(br-version)`. The {`SPC`} and {`DEL`} keys will then scroll the credits forward and backward, respectively, a windowful at a time.

Once an Environment has been loaded, entering and quitting the browser are rapid actions, so that you may smoothly transition between editing and browsing as needed. If you leave the browser using {`q`} and wish to browse the same Environment again, use {`C-u C-c o`}, which will immediately redisplay the browser just as you left it.

2.2 Displaying Top-Level Classes

The OO-Browser starts by displaying all classes in the Environment. The following commands filter the set of classes so that only *top-level classes*, those that do not inherit from any others, are shown. The browser can show all top-level classes or System or Library classes only. Once in the browser, use:

{s} to show only top-level System classes;

{l} to show only top-level Library classes;

{t} or {T} to show all top-level classes in the Environment.

The use of any of these commands does not affect the ancestry or descendancy trees for any given class. Each simply limits which trees are easily accessible for browsing. For example, selection of Library top-level classes only, followed by the browser show children command, {c} (br-children), would display the name of a System class if the System class directly inherits from the Library class.

To see an ordered listing of all of the classes in a particular part of an Environment, use a prefix argument with the commands given above:

{C-u s} shows all System classes;

{C-u l} shows all Library classes;

{C-u t} shows all Environment classes. {A} is a shortcut that does the same thing.

2.3 Moving to Entries

Many browser commands operate on the current entry of the selected listing window. {C-n} (br-next-entry) moves point to the next entry in a listing buffer. {C-p} (br-prev-entry) moves to the previous entry. Both take prefix arguments and use them as the number of entries by which to move.

2.4 Writing a Listing to a File

Many standard editing keys are rebound within listing buffers to issue browser-specific commands. Occasionally, you need to be able to store and to edit listing buffers. The {C-c C-w} (br-write-buffer) command provides this capability. The command prompts for a file name under which to save the current buffer. You may then quit the browser, read in the file and edit it as a plain text file.

2.5 Browsing Children and Parents

The {c} (br-children) command displays the children of the class at point. The {p} (br-parents) command displays the parents of the class at point. {C-u c} displays the children of all of the classes from the present listing window; {C-u p} does the same for parents.

2.6 Browsing Descendants and Ancestors

The OO-Browser is very fast at computing ancestor and descendant hierarchies, accounting for multiple inheritance and cycles where permitted. Descendant and ancestor listings provide an immediate overview of some key relationships among class groupings.

With point on any class entry line in a listing buffer, {d} (br-descendants) shows descendants for the class and {a} (br-ancestors) shows ancestors. {C-u d} shows the descendant trees for all of the classes in the current listing; {C-u a} does the same for ancestors.

The ancestor tree for a given root class is normally shown branching out from the root class. This means that higher-level ancestors, those further away from the root class, are shown in descending trees below lower-level ancestors. The leaves of the tree represent the ancestors furthest from the root, as you might expect.

This, however, is the inverse of inheritance trees. Some people prefer to see ancestor trees like inheritance trees, with parents above children. This is an *inverted ancestor tree*. To obtain this view of ancestors, use {`M- -1 a`} for ancestors of the current class. For ancestors of all classes in the current buffer, use {`M- -2 a`}, or any negative prefix argument less than -1. Inverted ancestor trees may be set as the default by making the variable `br-invert-ancestors` true: '`M-x set-variable RET br-invert-ancestors RET t RET`'. This is a personal setting that affects all Environments used by the browser.

2.7 Viewing and Editing

One of the major uses of the OO-Browser is to view or to edit class source texts. The {`v`} `(br-view-entry)` command will view the source for the class or element name (attribute, method or instance) at point in a read-only mode within the viewer window; it will not select the viewer window. The {`e`} `(br-edit-entry)` command performs a similar function, except that it edits the element source in a read-write mode, if the user has write permission for the source file. It also selects the viewer window.

A prefix argument to either of these commands, as in {`C-u v`} or {`C-u e`}, causes them to prompt for the entry to display. Full class and element name completion is provided once an Environment has been loaded and built. See Section 2.21 [Completing Names], page 17.

The value of the variable `br-edit-file-function` is the function that the browser calls when it displays a source file for editing. The value of `br-view-file-function` is the function called to view a source file. See Section 3.1 [Using an External Viewer or Editor], page 23, for information on using non-Emacs editors and viewers with the browser.

If you have no read access rights to a file, this will be apparent when the browser tries to display the file and fails. If you lack write permission to the class source file, the standard `br-edit-file-function` may display the file in a read-only mode (indicated by two percent signs, %%, at the front of the buffer modeline). This is a warning that you should not attempt to edit the file. In some cases, you may really need to edit such a file; then, you may toggle the buffer between read-only and read-write modes via the Emacs command, `(vc-toggle-read-only)`, usually bound to {`C-x C-q`}.

Once an entry has been displayed for viewing, {`SPC`} will scroll its source text up (forward) a windowful; {`DEL`} will scroll it down (backward) a windowful. In fact, this is a general means for scrolling the OO-Browser viewer window whenever point, as shown by the Emacs block cursor, is in a listing window.

For finer control over scrolling when in a listing window, use the {`.`} and {`,`} keys to scroll the viewer window by one line forward and backward, respectively. {`<`} will scroll the viewer buffer to its beginning; {`>`} will scroll display the end of buffer.

When point is moved to the viewer window, you must use {`C-v`} to scroll up and {`M-v`} to scroll down, assuming the standard Emacs key bindings are in use.

Sometimes one needs to quickly switch back and forth between the viewer window and the current listing window. The normal Emacs window movement commands are often

cumbersome in such instances. Instead use `(br-to-from-viewer)` bound to {`C-c C-v`}. This allows the desired back and forth movement. It acts as a toggle switch, alternately moving between the buffer in the viewer window and the prior listing buffer.

By default, the OO-Browser displays class definition files in their entirety. If there are multiple classes in a file, you will be able to scroll through all of them. If you prefer that only the selected class be visible, enable the `br-narrow-view-to-class` option flag. When set to '`t`', this flag narrows the source buffer so that only the class of interest and its preceding comments are visible. To examine other classes in the same file, you must execute a {`C-x n w`} `(widen)` command when in the narrowed buffer. (Use {`C-x w`} under Emacs 18.)

It may be helpful to use the full frame to view or edit a buffer of source code if the browser is run while using a small screen. If point is in a listing buffer, press {`1`}, the number one, to expand the viewer window to the dimensions of the full frame. When the browser is re-invoked, it will look just as it did before. If point is in the viewer window, {`C-x 1`} `(delete-other-windows)`, will do practically the same thing, except that when the browser is re-invoked it will not look precisely as it did before.

With point in a listing window, the buffer displayed in the viewer window may be killed with the {`C-c C-k`} `(br-kill)` command. (A killed buffer is removed from the current Emacs session.) With point in the viewer window, use the standard Emacs command {`C-x k`} `(kill-buffer)` instead.

2.8 Browsing Elements

A *feature* of a class is either a routine or attribute defined in the class. An *element* is either a feature or an instance of a class. An *instance* is an object whose type is that of a particular class, which therefore shares attributes and methods with the class.

Most OO-Browser languages support feature browsing, as documented in Chapter 6 [Languages], page 28. Instance browsing is supported in very limited form for class instances which exist within the code itself. For example, under Common Lisp and CLOS, a default class called `[function]` is defined whose instances are all named functions defined within the source code. A *default class* is a class automatically created by the OO-Browser to categorize standard elements of particular language Environments. The *instances* of default classes are constructs statically defined within Environment source code. Default classes themselves are not defined within the source code since they exist only to provide convenient categorization of constructs within the OO-Browser.

Use {`f`} `(br-features)` to display a listing of the features or elements of the class at point, including inherited features. For most languages, this includes all defined features but in some languages only routines are included. {`C-u f`} displays the features or elements of all classes in the present listing window.

Use {`r`} `(br-routines)` to display just the routines of the class at point. {`C-u r`} shows the routines in all listed classes. The {`=`} `(br-attributes)` command shows just the attributes of the class at point; each attribute entry is preceded by an `=` or an `&` character to distinguish them from other entry types. {`C-u =`}, as expected, applies to all listed classes.

By default, all inherited features, routines or attributes are shown by these commands, grouped together by the ancestral classes in which they are defined. (The OO-Browser does not yet distinguish between private and public attributes, so all attributes of ancestor classes will be shown, even if some are private.) Give any of these commands a zero prefix

argument, e.g. {M-0 f}, and they toggle the display of inherited features on and off for future listings. In fact, you may use the simpler key sequence, {0 f}, since the zero key is specially bound to serve as a prefix argument when in listing windows. If inherited features are off and there are no feature definitions for the class, the class definition is displayed so that you may browse its feature declarations.

Use {v} (br-view-entry) with point on an element name to view its source definition. Use {e} (br-edit-entry) instead to edit its source. Use {F} (br-feature-signature) to see the full signature tag of an element, which includes its argument names and types, if any. {C-u F} lists the signatures of all elements in the current listing. This is handy when several elements from the same class have the same name but differ in signature.

In languages that require method declarations separate from their definitions, some other keys are handy when browsing. (Presently these keys work only for C++ Environments.) Use {j} (br-feature-view-declaration) with point on a feature name to view its declaration; use {J} to edit its declaration within the viewer window.

See Section 2.14 [Using the Mouse], page 14, for how the context-sensitive Action and Assist Mouse Keys may be used for browsing elements.

2.9 Browsing Categories

The definition of a *category* is language-specific. Some languages such as Smalltalk use categories to group related classes together. The OO-Browser does not yet support this kind of category. It does support Objective-C categories, which segment each class into multiple groupings of related features. Objective-C class categories appear within parentheses when defined, so the OO-Browser displays category names with parentheses around them to distinguish them from classes. The aggregation of all of the categories defined by a class and its ancestors represents the complete class definition.

Use the {C} key (br-categories) with point on a class listing entry to obtain a list of the categories defined for the class within the Environment source code (this excludes inherited categories). Use {C-u C} to list the categories for all classes within the current listing. Thus, to see the full set of categories for a class, use {a} to list the ancestors of the current class and then {C-u C} to show all direct and inherited categories of the class.

Use {v} to view or {e} to edit the class category definition for the category entry at point. See Section 2.11 [Browsing Implementors], page 13, for an explanation of how to browse the classes that directly implement a category.

Use {f} with point on the default [category] class to list all categories defined in the Environment.

2.10 Browsing Protocols

The definition of a *protocol* is language-specific. It generally refers to an interface specification to which a class must conform. A class *conforms* to a protocol by implementing the set of features defined in the protocol interface. Protocols consist only of interfaces, without any method bodies, since conforming classes implement the necessary bodies. They generally differ from abstract classes, such as those found in Java and Eiffel, in that they define only the interface for a single facet of a class. The distinction is subtle, however, and abstract classes may be used as a substitute for protocols in languages that do not support protocols.

Presently, the OO-Browser supports protocols as a distinct construct under Objective-C and Java (where they are called *interfaces*). Objective-C protocols are sometimes called *formal protocols*. Protocol interfaces are specified in a manner similar to classes. A single protocol may inherit from any number of other protocols; therefore, any conforming class must conform to all of its ancestor protocols.

Objective-C class definitions reference within angled brackets the protocols to which they directly conform. Therefore, the OO-Browser displays protocol names within <angled brackets> to distinguish them from classes.

Commands that work on class listing entries generally also work on protocol entries, notably display of parents, children, ancestors or descendants. See Section 2.11 [Browsing Implementors], page 13, for an explanation of how to browse the classes that conform to a protocol.

By default, all protocols and classes referenced within an Environment are listed when the Environment is first loaded. Protocols are also included when the {`t`} command is used to show top-level classes only. If you prefer to omit protocols from such listings, use the toggle menu item `Options/List-Protocols-with-Classes` or the key sequence {`M-0 P`}; then issue the {`T`} or {`A`} commands to list top-level or all classes, respectively. The toggle menu item sets the value of the variable, `br-protocols-with-classes-flag` to '`t`' or '`nil`'.

If you have chosen to not list protocols with classes, you can still get a listing of protocols referenced within the Environment by moving point to the default class `[protocol]` or `[interface]` included in the top-level class listing of appropriate language Environments and then issuing the {`f`} command.

Use the {`P`} key `(br-protocols)` with point on a class listing entry to obtain a list of the protocols to which the class conforms (including inherited protocols). Use {`C-u P`} to list the protocols for all classes in the current listing. Use {`v`} or {`e`} when on a protocol entry to view or edit its definition. See Section 2.11 [Browsing Implementors], page 13, for a description of how to list the classes that physically define a protocol's methods (which is different than the set of classes that conforms to a protocol).

2.11 Browsing Implementors

Sometimes it is important to see the set of things that implement a feature, category or protocol. These are called *implementors*. With point on an element, category or protocol listing entry, {`I`} `(br-implementors)` will compute and display its list of implementors, which then may be browsed like any other listing entry. {`C-u I`} will do the same for entries in the present listing.

Move point to an implementor class name and then use {`v`} or {`e`} to view or edit the element associated with the class. If an element name is defined with different signatures in a single class, the class is listed as an implementor multiple times. Each class entry displays a different element. {`C-u F`} displays the element signatures for all of the class entries in the present listing buffer.

2.12 Exiting a Listing

When done with a browser listing buffer, exit from it with {`x`} `(br-exit-level)`. This command erases the current listing window and returns to the previous listing level, if any.

You may exit a single level at a time or all the way back to the top-level listing buffer by sending a prefix argument value to the command, {`C-u x`}.

There is no need to exit from listing buffers to quit from the browser. You may quit, perform other actions, and then re-invoke the browser at the same point from which you left. See the next manual section.

2.13 Quitting and Refreshing the OO-Browser

Use {`q`} (`br-quit`) to quit from the browser temporarily. The same command with a prefix argument quits from the browser permanently and kills all non-modified browser buffers. It will not kill any of the remaining class source buffers.

If you are familiar with Emacs windowing, you may quickly alter the window configuration of the browser frame, either intentionally or more likely unintentionally. If you execute non-browser Emacs commands while in the browser, you may find other buffers have taken the place of your browser buffers. In either case, you may refresh the browser display and restore it to the way it was when you originally invoked it, by using '`M-x br-refresh RET`' or with {`C-c C-r`} when in a browser listing buffer.

2.14 Using the Mouse

Once configured, mouse control within the OO-Browser is helpful and easy to use. Under InfoDock, XEmacs and Emacs 19 or higher, the right mouse button, called the Menu Key, pops up a menu of OO-Browser commands when clicked within an OO-Browser listing buffer. Under XEmacs and Emacs 19 or higher, the same menu is added to the menubar used in listing buffers. Under InfoDock with mode-specific menus turned on, the entire menubar is devoted to OO-Browser commands, with each submenu from the popup menu broken into a separate menubar entry. See Appendix B [Menus], page 41, for a detailed summary of the available menu commands.

Even if the above features are not available to you, if you have mouse support in your version of emacs, the following features are available. A single mouse button, called the *Action Key*, is used for most purposes such as displaying classes and features, as well as following references within code. Within listing buffers, the Action Key is bound to the middle mouse button on systems which typically have three mouse buttons and to the left mouse button on two-button systems. Use the OO-Browser `3-Button-Mouse` option to inform the browser that your system has three buttons, if it does and you are running under a Microsoft operating system (where systems typically have only two buttons).

Outside of listing buffers, the Action Key location depends on both the type of editor that you are using and the number of mouse buttons available. The Action Key is bound to the shift-middle mouse button under XEmacs or Emacs, to the middle mouse button under InfoDock, or to the shift-left button on a two-button mouse under any of these editors.

A second mouse button, called the *Assist Key*, is used for help and other ancillary functions. The Assist Key is bound to the shift-right button. Both of these keys perform context-sensitive actions based on where they are clicked. See Section 2.15 [Getting Help], page 15, for details on displaying a summary that describes their behavior in each context, any time you are using the browser. Below we discuss these behaviors.

Within an empty listing buffer, a click of the Action Key displays a buffer that summarizes the browser's key bindings (pressing {`h`} within any listing window displays this same

help buffer). We call this the *OO-Browser command menu* because an Action Key click within one of the curly brace delimited key sequences invokes the command bound to that key.

A click of the Assist Key within an empty listing buffer displays a menu of the existing source code buffers for the current OO-Browser Environment language, such as Java. Within this menu, the Action Key selects a buffer for display; the Assist Key marks the buffer for deletion. To perform the deletes, click the Action Key after the last line of the menu. If the Assist Key is clicked after the last line, the deletes are undone and a list of all current editor buffers is shown. This permits you to select buffers other than those containing classes.

The mouse buttons may be used to scroll the viewer window. An Action Key click at the end of a line scrolls that line to the top of the viewer window; the Assist Key puts the line clicked upon at the bottom of the window. This is called *proportional scrolling* since the amount of the scroll is relative to the position of the selected line.

When in an OO-Browser listing buffer, the Action Key acts as follows. If the key is pressed:

- on a blank line following all entries or in a blank listing buffer, the browser command help menu is displayed in the viewer window;
- on a default class name, the statically defined instances of the default class are listed;
- at the beginning of a (non-single character) class name, the class' ancestors are listed;
- at the end of an entry line, the listing is scrolled up proportionally so that the current line is at the top of the window;
- on the '...', following a class name, point is moved to the class descendency expansion;
- before an element entry, the element's implementors are listed;
- anywhere else on an entry line (i.e. on the entry), the entry's source is displayed for editing.

The Assist Key acts as follows when in a listing buffer. If it is pressed:

- in a blank buffer, a selection list of buffer files is displayed;
- on a default class name, the statically defined instances of the default class are listed;
- at the beginning of a (non-single character) class, the class' descendants are listed;
- at the end of an entry line, the listing is scrolled down proportionally so that the current line is at the bottom of the window;
- on the '...', following a class name, point is moved to the class expansion;
- anywhere else on a class line, the class' elements are listed;
- anywhere else on an element line, the element's implementors are listed;
- on a blank line following all entries, the current listing buffer is exited.

2.15 Getting Help

The OO-Browser is very intuitive to operate (much more so than it is to describe textually). However, help is always just a key or button press away when needed. Besides the online and printed versions of this manual, the command menu built-in to the browser and bound to {`h`} `(br-help)` serves as an online quick reference for key bindings and commands. The {`H`} `(br-help-ms)` command displays the `br-help-ms` file, a table that summarizes mouse

operations within each OO-Browser context. For more extensive documentation on each browser key, use the Emacs command {`C-h k <key-sequence>`}.

2.16 Locating Entries

The {`w`} `(br-where)` command locates the source file associated with a listing entry and displays it together with the entry name within the viewer window. A prefix argument as in, {`C-u w`}, causes the command to prompt for the class or element name to locate. Full completion is provided. See Section 2.21 [Completing Names], page 17.

The {`m`} `(br-match)` command offers a quick mechanism for locating any classes in the Environment whose names match to an expression in part or in whole. The browser prompts for the expression to use. All matching names are displayed in ascending order. By default, the expression is treated as a regular expression, i.e. a pattern match. A prefix argument sent to the command tells it to treat the expression as a string.

After each search, the command reports the number of matching classes found and displays them in the current listing window. It then prompts for another expression used to narrow the search further. This cycle continues until the {`RET`} key is pressed without entering an expression. This process allows for easy location of desired classes.

When the command is invoked (first time through the cycle), if the {`RET`} key is pressed without giving a match expression, the search will match to all classes referenced in the Environment.

If you want a regular expression to match to whole class names exclusively, begin it with a '`^`' and end it with a '$' character. These match to beginning of name and end of name, respectively. Thus, `"^....$"` matches to class names with exactly four characters. A string match always matches to any class name that contains the matching string.

2.17 Filtering Entries

The {`M`} `(br-match-entries)` command works much like the `(br-match)` command described in, Section 2.16 [Locating Entries], page 16, except that it matches only to entries in the current listing buffer. It thus allows you to filter a listing to just those entries that you care to browse. It prompts you for a regular expression of entries to match and then deletes entries that don't match. A prefix argument sent to the command tells it to treat the match expression as a string.

After each search, the command reports the number of matching entries found and displays them in the current listing window. It then prompts for another expression to match. The selected set is then filtered once again. This cycle continues until the {`RET`} is pressed without giving an expression. This process allows for easy incremental filtering of listings.

When the command is invoked (first time through the loop), if the {`RET`} key is pressed without giving a match expression, the search matches to all entries in the listing, so no filtering is done.

If you want a regular expression to match to whole entries exclusively, begin it with a '`^`' and end it with a '$' character. These match to beginning of line and end of line, respectively. Thus, `"^....$"` matches to entry lines with exactly four characters. A string match always matches to any entry that contains the matching string.

2.18 Ordering Entries

Once you have a desired set of names in a browser listing window, you may want to re-order them. For a simple ascending ordering by name, use {o} (br-order). To sort the lines in the current listing window accounting for leading whitespace, use a positive prefix argument. To sort the lines in descending order accounting for leading whitespace, use a negative prefix argument. You should note that all of the top-level class display commands automatically sort their output lists into ascending order. See Section 2.2 [Displaying Top-Level Classes], page 8.

To sort in descending order, first sort into ascending order with {o} to strip any leading whitespace and then use a negative prefix argument to sort the names into descending order.

2.19 Summarizing Environments and Classes

The {#} (br-count) command displays in the minibuffer the number of entries within the present listing buffer.

The {M-c} (br-class-stats) command displays in the minibuffer window the number of parents and children for the selected class; with a prefix argument, it prompts for the class name to use.

The {M-e} (br-env-stats) command displays the specification for the current Environment along with a few Environment statistics in the viewer window, namely: the OO-Browser version used to build the Environment, the start and end times of the most recent build of theis Environment, total classes, number of System and Library classes, and the number of duplicate and undefined classes. This command also displays version information on the editor and some of its tools.

With a prefix argument, {M-e} displays in the minibuffer window the basic statistics only, leaving the contents of the viewer window intact.

2.20 Deleting Classes

To delete a class from the Environment, display the class name in a listing window using the {m} (br-match) command, if necessary. See Section 2.16 [Locating Entries], page 16. Then move point to the desired class name and press {C-c C-d} (br-delete) to delete the class. This will remove the class name at point and will delete the class from the Environment.

2.21 Completing Names

Whenever the browser prompts for a name and an Environment has already been loaded or built, you can use the browser's name completion facilities to help you enter the name. These features let you type as much of the name as you know and then have the browser fill in what it can. The relevant keys are:

{TAB} complete as much as possible of a class or element name;

{SPC} complete up to one word of a class or element name;

{?} show all possible completions for a class or element name.

You can also use the browser's completion facilities outside of the browser, for example, when editing code. See Chapter 5 [Using Standalone OO-Browser Features], page 26, and the documentation produced from typing '`C-h f br-complete-symbol RET`'.

2.22 Browsing Graphically

The X interface to the OO-Browser is named, *Xoobr*, and is pronounced ex-owe-owe-browse-er. It provides a simple but effective means of navigating through hierarchy and element relations.

Windows-specific versions of the OO-Browser now include a native Windows graphical browser that works the same way as the Xoobr described in this section but is simply named *oobr.exe*. Use this section as a reference for either version of the graphical browser.

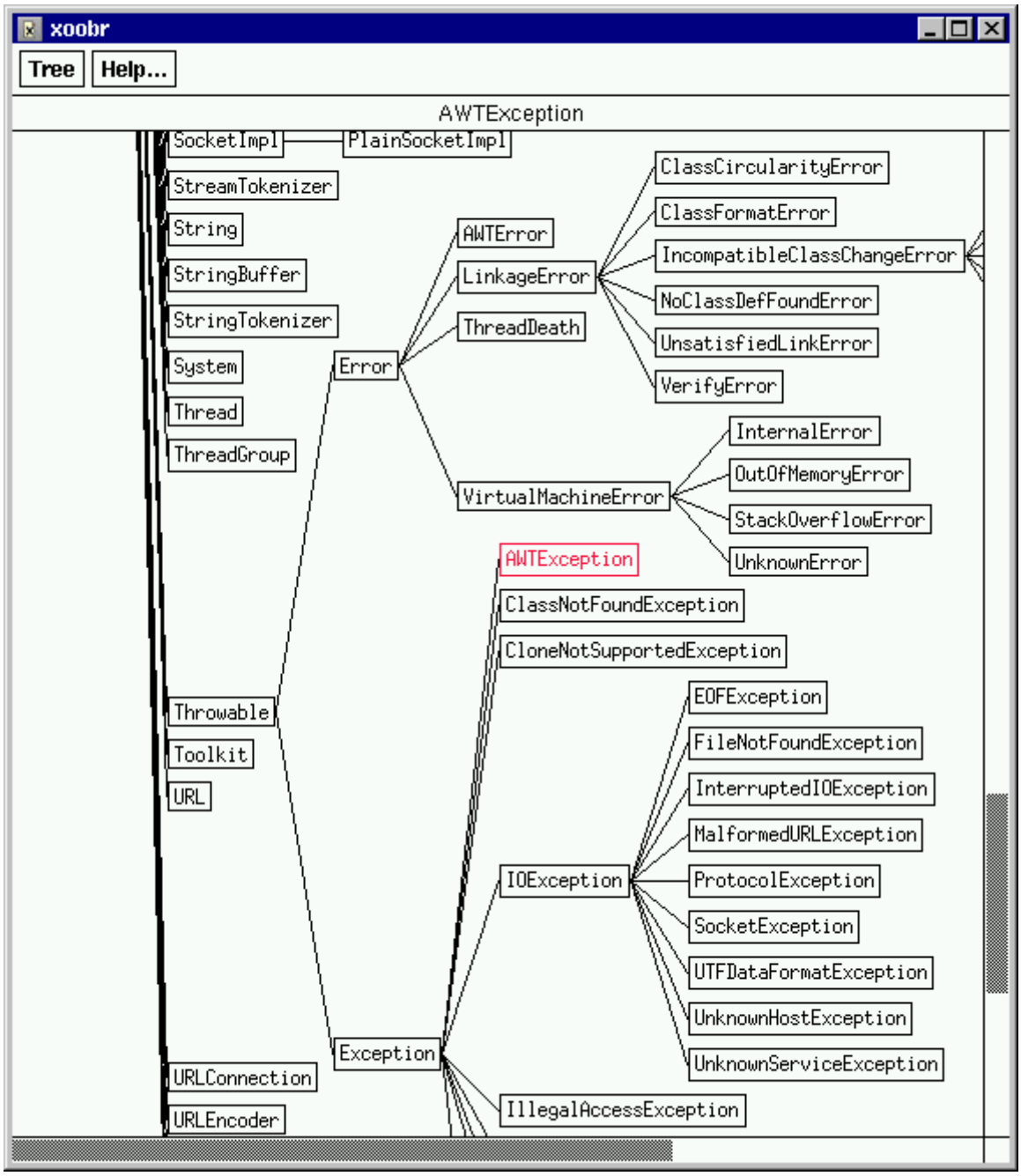


Image 2.1: The X OO-Browser Screenshot

Any number of Xoobr sessions may be established at the same time to yield different views over Environments. Each session may show relations from a different Environment (even a different language) than the others do. The textual OO-Browser is used to select the set of classes for display in an Xoobr session. For this reason, Xoobr is almost always

invoked from within the textual OO-Browser. The following keybindings are all used within the textual browser to manage Xoobr views.

{`M-d`} `(br-tree)` selects the current class and displays its descendancy graph in tree-form by starting a new Xoobr session. With a prefix argument, {`C-u M-d`}, it displays descendancy trees for all classes within the current browser listing. They are all grouped under an imaginary root node so as to maintain the concept of one tree per Xoobr view.

Use {`M-f`} `(br-tree-features-toggle)` or the `Options/Graphical-Descendant-Features` menu item, to set whether or not the features of listing classes are shown as child nodes in any graphical descendency views created by the {`M-d`} command. (The setting applies across all OO-Browser languages. The default setting is not to add features to Xoobr views.)

{`M-g`} `(br-tree-graph)` displays the current listing buffer's entries in a graphical form. It ignores the add features setting so that you can capture the current listing without the need to alter that setting.

The {`M-k`} `(br-tree-kill)` command prompts to see if you want to terminate all Xoobr sessions started from within the current editor session. If you answer affirmatively, all such processes disappear, as your screen will quickly indicate.

(The rest of this section discusses the user interface of the Xoobr.) Xoobr views are meant to complement the textual browser interface. Therefore, the two most common actions used in the text browser are performed in a similar manner within an Xoobr view. A click on a node with the left mouse button displays the appropriate class text in the user-selected editor, ready for editing. See Section 3.1 [Using an External Viewer or Editor], page 23. A click of the middle button performs similarly but displays the associated class for viewing only.

The right mouse button when depressed over an Xoobr node displays a short menu of commands that may be applied to the node. The only ones of real interest at this point are the collapse and expand entries which let you hide and then restore the display of a node's subtree. This yields precise control over the amount of detail you receive in various parts of the hierarchy.

The Help button in the Xoobr menubar displays a few pages of help text regarding the program itself.

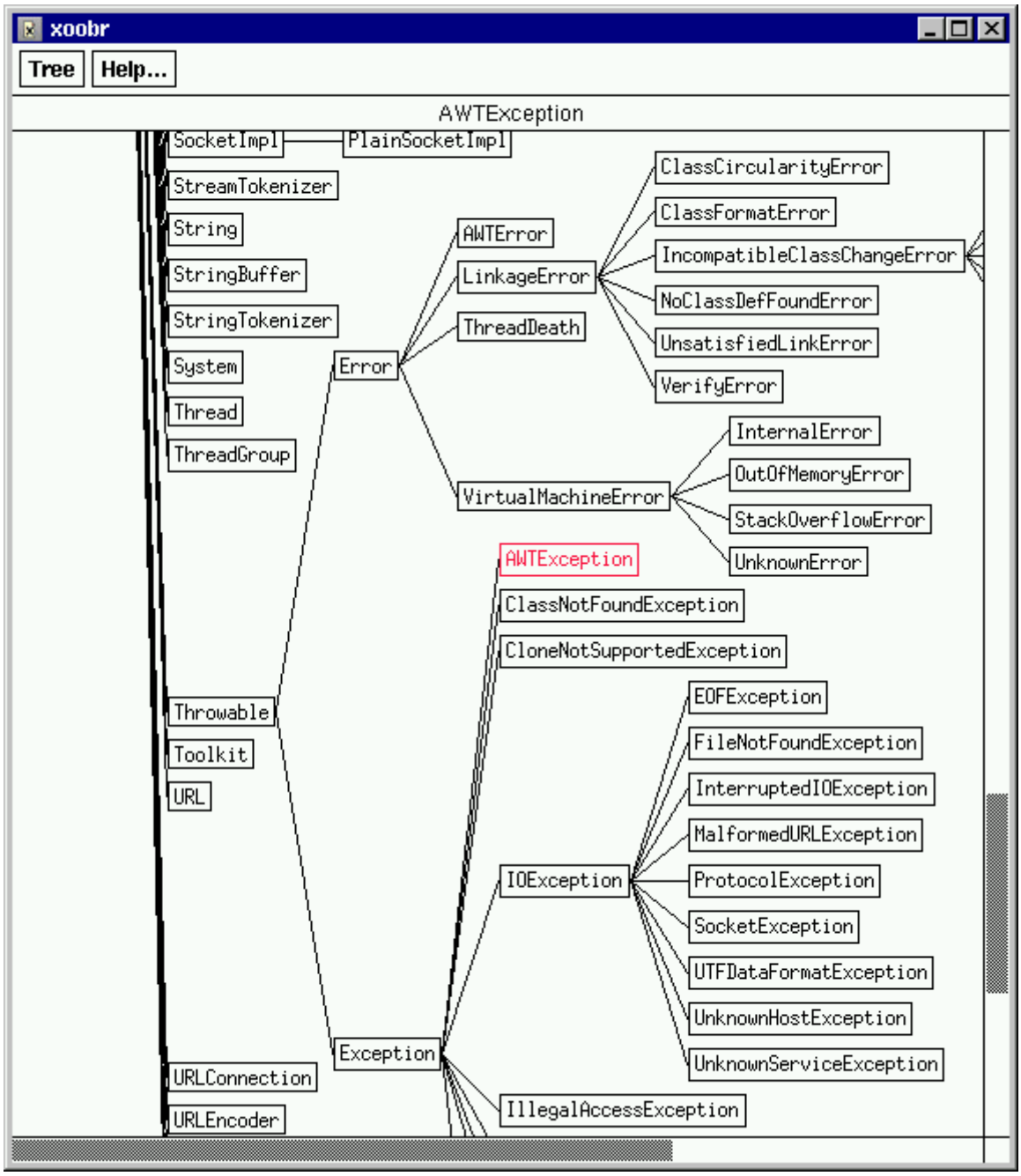


Image 2.2: The X OO-Browser Screenshot

The Xoobr menubar may be useful in the future but should generally be ignored for now. The only menu entry of any import is found under the file menu, labeled `Quit`. Xoobr processes may also be terminated by issuing the kill command mentioned previously. A third means of killing such processes is by sending the permanent `(br-quit)` command,

{`C-u q`}, to the textual browser. You will then be prompted as to whether to terminate all Xoobr sessions started from within the current editor session.

3 OO-Browser Options

This chapter explains how to set a variety of OO-Browser configuration options. A number of sections call for adding a line or two of variable settings to your personal initialization file in order to make a permanent change to an OO-Browser option. If you use InfoDock, the preferred initialization file is `~/.infodock`. If you use Emacs or XEmacs, use `~/.emacs` instead.

3.1 Using an External Viewer or Editor

The OO-Browser lets you select your desired editor and viewer programs when you use a multi-windowed display. By default, both of these tasks are handled by your InfoDock or Emacs editor so that the browser will work when run on dumb terminals. If you choose an external editor or viewer for use under a window system, InfoDock or Emacs will still automatically be used whenever you invoke the browser from a dumb terminal.

The `br-editor-cmd` variable is used to set the command that invokes the editor whenever source code is to be displayed by the OO-Browser. Arguments to the command should be placed in the variables named `br-ed[1-9]`, with one string argument per variable. Unused variables should have the value '`nil`'. Bear in mind that the command must generate a new window under your window system. For example, the vi editor under UNIX does not create its own window, it runs within the window in which it is created. Under X one would create a new xterm window and then invoke vi. The command line would be, '`xterm -e vi`'. The settings in your personal initialization file would look like this.

```
(setq br-editor-cmd "xterm" br-ed1 "-e" br-ed2 "vi"
      br-ed3 nil br-ed4 nil br-ed5 nil
      br-ed6 nil br-ed7 nil br-ed8 nil br-ed9 nil)
```

If you want to view classes in a read-only fashion with an external viewer, set the following `br-viewer-cmd` and `br-vw[1-9]` variables in the same way as for the editor variables above.

For example, to use `xmore`, an X-compatible version of `more`, as your viewer, use the following settings (assuming all the `br-vw` variables are already '`nil`').

```
(setq br-viewer-cmd "xmore")
```

The OO-Browser now includes an OO-Browser menu item, `Options/Use-Vi-as-Editor`, that enables the use of '`vi`' as the editor and '`more`' as the viewer. Simply turn this option on to get this behavior. Follow the prior instructions to make such a setting permanent.

3.2 Toggling Inherited Features Display

By default, when the OO-Browser lists features of a class, it shows both the ones lexically defined within the class source text and the ones inherited from ancestor classes. Each feature is listed below the class in which it is originally defined, for clarity. Sometimes it is helpful to see only the lexically defined features of a class. In such cases, the menu item, `Options/Show-Inherited-Features`, toggles this setting. If you want this off by default, you may add the following line to a personal initialization file.

```
(setq br-inherited-features-flag nil)
```

3.3 Adding Features to a Graphical View

See Section 2.22 [Browsing Graphically], page 18.

3.4 Keeping Viewed Classes

The `br-keep-viewed-classes` flag is turned off by default, which means that each class buffer read in by the browser will be deleted when the next one is browsed. If it is set to '`t`', all viewed classes are left around for later selection.

In typical use, the burden of having to manage all viewed classes is greater than the benefit of leaving them in memory. This is why the flag is off by default. If you choose to leave them around, the class buffer menu can be used to delete buffers when you want to trim down the number of them. See Section 2.14 [Using the Mouse], page 14, for details on this technique.

The value of the `br-keep-viewed-classes` flag may be toggled with the `(br-toggle-keep-viewed)` command or with the menu item, `Options/Keep-Viewed-Classes`.

3.5 Inverting Ancestor Trees

Ancestor trees are normally shown to emphasize how the trees branch out from the current class, with the most distant ancestors shown as leaves of the ancestry tree. If you prefer that all listing displays reflect the class inheritance structure, with children below parents, you may invert the ancestry tree listings by adding the following line to your personal initialization file.

```
(setq br-invert-ancestors t)
```

This will make the most distant ancestors appear as roots of the trees and parents (the nearest ancestors) appear as leaves, since they inherit from the higher level ancestors. This is a global OO-Browser option, it affects all Environments.

4 Personal Customization

The `br-skip-dir-regexps` variable is a list of regular expressions which match directory names that the OO-Browser will not descend when scanning source code trees. By default, it skips RCS, SCCS and Eiffel .E subdirectories.

The `br-file-dir-regexp` variable is a regular expression that matches to file and directory names that the OO-Browser should scan. Any others, including those matched by `br-skip-dir-regexps`, are ignored.

The following hook variables are supplied as a means of customizing the startup procedure of the OO-Browser. Set them as you would any other Emacs Lisp hook variables in your personal initialization file. They all do nothing by default.

If you want a set of actions to occur each time after the OO-Browser is invoked, attach them to `br-mode-hook`. After `br-mode-hook` is evaluated, a language-specific hook is also run, for the setup of language-specific options associated with the current Environment. The names of these hooks are as follows:

C++ and C
: `br-c++-mode-hook`

CLOS
: `br-clos-mode-hook`

Eiffel
: `br-eif-mode-hook`

Java
: `br-java-mode-hook`

Objective-C
: `br-objc-mode-hook`

Python
: `br-python-mode-hook`

Smalltalk
: `br-smt-mode-hook`.

Finally, if you want a set of actions to occur each time after a new browser listing buffer is created, set `br-class-list-hook`.

5 Using Standalone OO-Browser Features

A number of browser features may be used independently of the browser user interface. First, an Environment must be selected and loaded into the OO-Browser via {`C-c o`} or '`M-x oo-browser RET`'. When the browser user interface is displayed, use {`q`} to quit.

Alternatively, an Environment may be loaded without invoking the browser user interface by using '`M-x br-env-load RET`'. The standalone browser features will use the newly loaded Environment.

(Note that terms in all CAPITALS below are an ordered set of parameters prompted for and used by each command. The key bindings are set up in the file `br-site.el` for each OO-Browser language that you purchased.)

The following commands are available for standalone use.

`br-feature-edit-declaration {C-c M-j}`
: Prompts with completion for a CLASS::FEATURE argument and then edits the associated declaration. If point is on a feature definition signature in a code buffer (prior to any of its arguments), the default is to edit that feature's declaration. An error is signaled if the declaration is not found. Presently, this command is available under C++ only.

`br-find {C-c M-f}`
: Prompts with completion for a class or element name from the current Environment and displays its definition for editing.

  During code edits and debugging sessions, it is often helpful to be able to look at a class or element definition without having to invoke the browser to locate it. The `(br-find)` command does exactly that by prompting for a name and then displaying the named class or element definition.

`br-find-class`
: Displays the source text for a class matching CLASS-NAME in VIEW-ONLY mode (or edit mode if '`nil`').

  The `(br-find-class)` command is similar to `(br-find)`. It prompts for a class name and then displays its source file in a viewable, read-only mode. To display a class file in an editable mode, send a prefix argument to this command.

`br-complete-symbol {M-TAB}`
: Performs in-buffer completion of a type or element identifier before point.

  When writing code and entering class attribute definitions (variable definitions), you often have to enter class names repetitively. The `(br-complete-symbol)` command completes and inserts a class name at point within the current buffer. The following example illustrates its usage.

  my_list: LIN<– (point is here)

  {`M-TAB`} is pressed:

  my_list: LINKED_LIST

`br-where {C-c M-w}`
: Prompts for a class or element name and displays the full path to its definition in another window. With an optional prefix argument, it inserts the path at point.

All of these commands offer full completion. See Section 2.21 [Completing Names], page 17, based upon the current OO-Browser Environment.

See Section 6.4 [Eiffel Specifics], page 31, for an Eiffel-specific standalone browser feature.

6 Language-Specific Notes

This chapter summarizes OO-Browser details that are specific to each language. There are no Smalltalk specifics.

6.1 C Specifics

The `br-c-tags-flag` variable controls whether or not C constructs are included within C and C-based language Environments. By default, this flag is true except on operating systems where the OO-Browser cannot find a UNIX-type shell such as '`sh`' or '`csh`'. In those cases, C constructs cannot be categorized by the OO-Browser. Use '`M-x br-toggle-c-tags RET`' to toggle the value of this variable. If you set it false before building an Environment, then C constructs will not be included in the Environment.

If you wish to build an Environment whose source code is entirely C, ensure that the `br-c-tags-flag` is enabled and then select C++ when asked for the language to browse.

C constructs are grouped into default classes for browsing. The elements of each default class are the instances of the associated construct within the Environment. Use normal element/feature commands to browse each instance.

```
DEFAULT CLASS    C CONSTRUCT
--------------------------------------
[constant]       #define constant
[enumeration]    enum {}
[enum_label]     label_name
[function]       non-member function()
[macro]          #define macro()
[structure]      struct {}
[type]           typedef {}
[union]          union {}
[variable]       <type> global_variable_name;
```

Within code that supports C constructs, an Action Key click on a reference to a global C identifier will display the identifier's definition.

6.2 C++ Specifics

See Section 6.1 [C Specifics], page 28, for details on C-specific support within C++ Environments.

6.2.1 C++ Listing Entries

C++ entities are categorized when shown within OO-Browser listing buffers. Classes are shown by name, without any prefix. Features are shown by name but are preceded by a special character that indicates the kind of feature. The following table describes each prefix:

`>` precedes pure virtual functions, which are specified but not defined within this class;

`%` precedes friend functions and classes, which have access to the class' private parts;

`+` precedes constructors and destructor functions;

`-` precedes any other type of member function;

`&` precedes static attributes (one attribute per class);

`=` precedes instance attributes (one attribute per object instance).

The following paragraphs discuss each of these types of features in more detail.

C++ pure virtual function declarations, which specify method interfaces but no implementations, appear in class feature listings. Their function names are preceded by the `"> "` string to identify them as pure virtual (deferred) functions. Pure virtuals may be treated like any other member functions within the browser. Since they lack definitions within their base classes, their declarations are shown when a view or edit command is given.

Friend functions and friend classes give members of another class access to the private parts of their class. Friend functions and classes appear in class feature listings preceded by the `"% "` string. The keys, {`v`} or {`e`}, display the friend declaration within the current class. Use {`V`} `(br-view-friend)` in a listing window to view the definition of the friend itself. If you use {`e`}, which leaves point on the friend declaration, a press of the Action Key will then move to the definition of the friend itself.

Methods and operators for creating and destroying objects are preceded by the `"+ "` string in listing buffers. All other method listing entries are preceded by the `"- "` string.

Static attributes (data members instantiated once per class) are shown preceded by the `"& "` string regardless of their types. Instance attributes (data members instantiated at each object creation) are shown preceded by the `"= "` string regardless of their types.

6.2.2 Source Code Element Selection

Once you have loaded an Environment, whether or not the OO-Browser user interface is on screen, you can use the Action Key to follow a variety of references within the code. Simply press the Action Key on an identifier or C++ construct and its associated definition or list of possible definitions or its declaration (if no definitions exist within the Environment) will be displayed for you to inspect or edit. More specifically:

- You can jump from a C++ class or method declaration to its definition by pressing the Action Key within the declaration signature (with point prior to its argument list). A press of the Action Key on a method definition signature (with point prior to its argument list) will also jump to the associated declaration so that you can pop back and forth between the two (assuming their signatures match up). If the method is defined within an ancestor class, a message at the bottom of the frame will announce the defining class whenever its definition is displayed. (Note that the {`j`} key can be used within browser listing buffers to view class and feature declarations. The {`J`} key works the same way but displays the entry for editing rather than viewing.)

  One stylistic tip: Each feature declaration should be terminated with a semicolon, rather than declared in a list, to ensure accurate scanning by the OO-Browser, i.e. don't use declarations like: `float f1, f2, f3;`.

  When in a code buffer, {`C-c M-j`} `(br-feature-edit-declaration)` will prompt for a feature name and will then display its declaration for editing. The key, {`C-c M-f`} `(br-find)` displays works similarly but displays definitions instead. See Chapter 5 [Using Standalone OO-Browser Features], page 26, for more complete operational descriptions of these commands.

- Classes can be browsed by pressing the Action Key on their names in inheritance clauses, feature signature argument lists or within scoped member references. It is therefore important to click on the method name part of a declaration when that is the element desired.
- C++ method, function call and single-level attribute reference browsing is now supported by pressing the Action Key on the member name of a call. The OO-Browser deals with much of the complexity of the C++ calling syntax so you need not (though it doesn't yet account for method overloading). In cases where it cannot determine a unique definition (e.g. where dynamic binding is involved), it pops up a list of possible definitions (method signatures). An Action Key click on any of these (or on the class names separating these signature lines) will display the definition within the source code. If the browser determines a unique base class of the call, it produces an ancestry tree of classes and intersperses the possible matching method signatures indicating the base class of each method in a fashion similar to how inherited features are shown in browser listing buffers.

  If no definitions are found, the browser tries to display a matching declaration. See Section 2.8 [Browsing Elements], page 11.
- You can browse include files by selecting their inclusion declarations (#include ...) within a source file. Include files delimited by double quotes are searched for in the following places: first, the directory of the current source file, then the directories listed in the variable `c++-include-path`, and then in any directories included in the current environment.

  Include files delimited by <angled brackets> are searched for in the following places: first, the directories listed in the variable `c++-include-path`, then the directories listed in the variable `c++-cpp-include-path`, and then in any directories included in the current environment. The variable `c++-cpp-include-path` should be set to a list of the standard directories searched by your C++ pre-processor. Each directory entry must end with a directory separator. On UNIX systems, this is the '`/`' character.

6.2.3 C++ Settings

By default, '`class`', '`struct`' and '`union`' definitions all constitute class definitions to the C++ OO-Browser. If you prefer some other criteria, you will need to modify the definition of `c++-class-keyword` in the `br-c++.el` file.

If you find that the browser is not scanning some of your C++ source files, you may be using file suffixes which it does not recognize. Examine the value of `c++-src-file-regexp` and add to it any special file suffixes that you use.

6.3 CLOS Specifics

6.3.1 Method Handling

In CLOS, methods may have explicitly typed parameters of the form, '`(<parameter-name> <class>)`'; these are called *specialized parameters*. Each such parameter defines the method within '`<class>`'. Thus, a single method definition can generate methods associated with many classes. The OO-Browser lets you navigate to a method definition from any of the

classes for which it is defined, since a method is included as an element of each of its constituent classes.

CLOS permits compile-time computation of the '`<class>`' of a specialized parameter through use of the expression, '`(eql <form>)`'. Since the OO-Browser cannot perform this computation, it treats such a parameter as non-specialized.

Methods may also contain non-specialized parameters of the form, '`<parameter-name>`'. The type of each such parameter defaults to the built-in root class, '`t`'. But a method is included in the class '`t`' by the OO-Browser only if none of its parameters are specialized. Otherwise, methods with more specific types which happened to include one or more non-specialized parameters would appear to not have a more specific type than '`t`'.

6.3.2 CLOS Settings

The OO-Browser automatically works with CLOS class and method definitions. But Lisp contains many other standard object types such as functions, macros and variables which you may wish to browse. These are handled through a configuration variable as explained below.

The `clos-element-type-alist` variable determines the Lisp definition constructs that the browser looks for and the associated default class under which instances of each construct type are grouped. Each element in the association list is a dotted pair whose first part is the function name string that defines an instance of a particular type, e.g. '`"defun"`'.

The second part is a default class name, a string, under which to assign each instance of the type, e.g. '`"function"`'. The OO-Browser always displays default class names with square brackets around them, e.g. '`[function]`', to distinguish them from classes defined within the Environment.

Here is the standard value of `clos-element-type-alist`.

```
((("defconstant"  . "constant")
 ("defconst"     . "constant")
 ("defun"        . "function")
 ("defgeneric"   . "generic")
 ("defmacro"     . "macro")
 ("defpackage"   . "package")
 ("defparameter" . "parameter")
 ("defsetf"      . "setfunction")
 ("defstruct"    . "structure")
 ("deftype"      . "type")
 ("defvar"       . "variable")
 ("fset"         . "function"))
```

The `clos-def-form-with-args-regexp` is a regular expression which includes a subset of the first items from the dotted pairs of `clos-element-type-alist`, namely those which require an argument list to uniquely distinguish them from other elements, e.g. functions.

6.4 Eiffel Specifics

Eiffel support has now been updated to Eiffel version 3, to the best of our knowledge. If you find any problems, please report them to <bug-oo-browser@gnu.org>.

6.4.1 Eiffel Listings

Eiffel entities are categorized when shown within OO-Browser listing buffers. Classes are shown by name, without any prefix. Features are shown by name but are preceded by a special character that indicates the kind of feature. The following table describes each prefix:

`-` precedes regular routines;

`=` precedes attributes;

`1` precedes once routines, which create shared objects;

`>` precedes deferred features, which are specified but not defined within this class;

`/` precedes external features, which are defined in a non-Eiffel language.

A detailed summary of a class may be generated with point on a class entry by pressing {`i`} (`br-entry-info`). By default, this shows a summary including the parents, attributes, routines and routine call summaries of the class. The command '`M-x eif-info-use-short RET`' instead causes the {`i`} key to run the Eiffel '`short`' command on the class, thereby displaying its specification. The command '`M-x eif-info-use-flat RET`' enables use of the '`flat`' command to the complete feature set of a class. '`M-x eif-info-use-calls RET`' resets the key to generate default summaries once again.

6.4.2 Source Code Element Selection

You can jump from an Eiffel element reference to its definition by clicking the Action Key on the reference. Selection of a feature name in an export clause displays the feature definition, even if it is renamed several times within ancestors. Parent classes may be browsed in a similar manner by clicking on their names in an inheritance or declaration clause.

The following example of locating a renamed feature is taken from an actual set of Eiffel library classes:

```
The user selects feature `subwindow' of POPUP_MENU
   inherited from WINDOW which renames `child' to `subwindow'
      inherited from TWO_WAY_TREE which renames `active' to `child'
         inherited from TWO_WAY_LIST
            inherited from LINKED_LIST which defines `active'.
```

The browser displays the feature `active` and explain to the user that feature `subwindow` of class `POPUP_MENU` is inherited from feature `active` of class `LINKED_LIST`. Location of this sort of feature definition would be incredibly tedious without programmatic support.

The algorithm used to locate features is dynamic, so if another class is inserted into the inheritance structure given above, the feature definition will still be found.

6.4.3 Eiffel Settings

Emacs has a feature called error parsing which lets you quickly jump to the line of code that supposedly triggered an error. See Section "Compilation Errors" in *the XEmacs Manual*, for information on error parsing. Some object-oriented language compilers display the name of the class and line number with each error message but do not include the filename containing the class source code. Emacs then has no way of parsing the error message. The browser class location facilities enable you to perform error parsing across any set of classes in a single Environment, given only this type of message. Interactive Software Engineering's

Eiffel compiler is an example of a compiler that (at least at one time) produced this type of error. The code for parsing ISE Eiffel error messages is included in `eif-ise-err.el`.

6.5 Java Specifics

The OO-Browser browses Java classes, interfaces, methods, and attributes. Earlier sections of this manual explain how to browse these entities. This section documents Java language specifics, including variable settings.

By default, Java Environments also include C constructs encountered when building each Environment. See Section 6.1 [C Specifics], page 28, for details on C-specific support within Java Environments.

Java entities are categorized when shown within OO-Browser listing buffers. Classes are shown by name, without any prefix. Interfaces are delimited by <angled brackets>. The following table describes the prefixes that preced each feature listing entry:

`=` precedes attributes;

`-` precedes regular methods;

`+` precedes methods associated with creating and destroying objects;

`>` precedes abstract method declarations; Abstract (deferred) methods may be treated like any other methods within the browser. Since there is no definition for such methods, their declarations are shown whenever an edit or view request is issued.

`/` precedes native methods, to indicate that such methods are divided between Java and another language; Native methods are like abstract methods in that only their interfaces are specified in Java. Unlike abstract methods, their method bodies are defined in external languages such as C to allow for machine-specific dependencies.

Java interface specifications, which define formal protocols to which classes must conform, are treated just like class definitions in browser listings. All the methods of such interfaces are abstract, however.

6.5.1 Java Interfaces

A *Java interface* specifies a formal protocol (set of method signatures and attributes) to which a class (the implementor) that uses the interface must conform. A class conforms to the interface by inheriting its attributes and by implementing the method bodies whose calling signatures are given by the interface. The class' descendants automatically conform to the interface through inheritance; they are considered descendants of the interface but not implementors of it. One interface may inherit from a list of other interfaces and thereby expand the set of methods which a conforming class must implement.

The OO-Browser can list and browse the source for:

the interfaces to which a class or an interface conforms
(press {`P`} on a class or interface listing entry);

all interfaces in an Environment
(press {`f`} on the [interface] default class entry);

the implementors of a interface
(press {`I`} on a interface listing entry).

See Section 2.10 [Browsing Protocols], page 12, for more details.

6.5.2 Source Code Element Selection

You can jump from a Java class method declaration to its definition by clicking the Action Key when within the declaration. If a method is inherited from another class, a message at the bottom of the frame will announce the defining class whenever its definition is requested.

Parent classes and interfaces may be browsed in a similar manner by clicking on their names in an inheritance or declaration clause. It is therefore important to click on the method name part of a declaration when that is the element desired.

6.5.3 Java Settings

The regular expression `java-class-keyword` defined in `br-java.el` determines which keywords are used to locate class and interface definitions within the Java OO-Browser.

If you find that the browser is not scanning some of your Java source files, you may be using file suffixes which it does not recognize. Examine the value of `java-src-file-regexp` and add to it any special file suffixes that you use.

6.6 Objective-C Specifics

See Section 6.1 [C Specifics], page 28, for details on C-specific support within Objective-C Environments.

The OO-Browser browses Objective-C classes, methods, categories and formal protocols. Earlier sections of this manual explain how to browse these entities. This section documents Objective-C language specifics, including variable settings.

Objective-C entities are categorized when shown within OO-Browser listing buffers. Classes are shown by name, without any prefix. Categories are delimited by (parentheses), while protocols are delimited by <angled brackets>. Methods are shown by name but are preceded by a special character that indicates the kind of method. The following table describes each prefix:

`-` precedes instance methods;

`+` precedes class (factory) methods that affect the factory object for the class.

6.6.1 Objective-C Categories

An *Objective-C category* is an internal class grouping that specifies and implements a set of related class features. The aggregation of all of the categories defined by a class and its ancestors represents the complete class definition.

The OO-Browser can list and browse the source for:

the categories of a class
(press {`C`} on a class listing entry);

all class categories in an Environment
(press {`f`} on the [category] default class entry);

the classes which implement a category
(press {`I`} on a category listing entry).

See Section 2.9 [Browsing Categories], page 12, for more details.

6.6.2 Objective-C Protocols

An *Objective-C protocol* is an interface specification (set of method signatures) to which a class (the implementor) that uses the protocol must conform. A class conforms to the protocol by implementing the method bodies whose calling signatures are given by the interface. The class' descendants automatically conform to the protocol through inheritance; they are considered descendants of the protocol but not implementors of it. One protocol may inherit from a list of other protocols and thus expand the set of methods which a conforming class must implement.

The OO-Browser can list and browse the source for:

the protocols to which a class or a protocol conforms
(press {`P`} on a class or protocol listing entry);

all protocols in an Environment
(press {`f`} on the [protocol] default class entry);

the implementors of a protocol
(press {`I`} on a protocol listing entry).

See Section 2.10 [Browsing Protocols], page 12, for more details.

6.6.3 Source Code Element Selection

You can jump from an Objective-C class method declaration to its definition by clicking the Action Key when within the declaration. If a method is inherited from another class, a message at the bottom of the frame will announce the defining class whenever its definition is requested.

Parent classes and protocols may be browsed in a similar manner by clicking on their names in an inheritance or declaration clause. It is therefore important to click on the method name part of a declaration when that is the element desired.

Include files may be browsed by selecting their inclusion declarations (#import ...) or (#include ...) within a source file. Include files delimited by double quotes are searched for in the following places: first, the directory of the current source file, then the directories listed in the variable `objc-include-path`, and then in any directories included in the current environment.

Include files delimited by angled brackets are searched for in the following places: first, the directories listed in the variable `objc-include-path`, then the directories listed in the variable `objc-cpp-include-path`, and then in any directories included in the current environment. The variable `objc-cpp-include-path` should be set to a list of the standard directories searched by your Objective-C pre-processor. Each directory entry must end with a directory separator. On UNIX systems, this is the '/' character.

6.6.4 Objective-C Settings

The regular expression `objc-class-keyword` defined in `br-objc.el` determines which keywords are used to locate class and protocol definitions within the Objective-C OO-Browser.

If you find that the browser is not scanning some of your Objective-C source files, you may be using file suffixes which it does not recognize. Examine the value of `objc-src-file-regexp` and add to it any special file suffixes that you use.

6.7 Python Specifics

See Section 6.1 [C Specifics], page 28, for details on C-specific support within Python Environments.

The OO-Browser can list and browse the source for the following Python constructs:

packages (press {`f`} on the [package] default class entry to list all of the packages in the Environment, each of which is preceded by an @ symbol)

modules (press {`f`} on the [module] default class entry to list all of the modules in the Environment)

classes (press {`f`} on a class entry to see its methods; there is no support for browsing its instance variables except by viewing the class' __init__ method definition)

functions (press {`f`} on the [function] default class entry to list the non-member functions defined in the Environment; this includes C functions)

globals (press {`f`} on the [global] default class entry to list all of the globals defined in the Environment)

Note that nested classes are not recognized by the browser, nor are run-time modifications to classes.

Documentation for the Python listing entry on the current line may be displayed with the {`i`} `(br-entry-info)` command. This displays documentation for packages, modules, classes, functions and methods. It utilizes the `pydoc.el` interface to the Python pydoc library, when available, to provide extended documentation but it will nevertheless display basic documentation strings if this package is not installed.

An Action Key press on a Python `import` or `from` statement will display the source of the item under point, which may be a module, class, method, function or variable name, if the source is found within the current Environment. First, the module is searched for within the current Environment directories. If it is not found, it is searched for within the paths listed in the variable, `python-import-path`, which is initialized to the value of the *PYTHONPATH* environment variable or if that is not defined, to `/usr/local/lib/python`.

Appendix A Glossary

Concepts pertinent to operational usage of the OO-Browser are defined here. If some GNU Emacs terms are unfamiliar to you, see *[Stallman 93]*.

`Ancestors`
: All classes above a class in the inheritance hierarchy.

`Attribute`
: A data item declared with a class. Most attributes are instance-specific; each instance object of a class has its own copy of the attribute so that it can maintain a separate state. Some languages allow for class attributes where all instances of the class share one copy of the attribute and thereby maintain shared state.

`Category`
: Under most languages, a logical grouping of related classes. The OO-Browser does not yet have any support for this kind of category.

  Under Objective-C, a category is a partial class definition that implements a related set of methods. The full class definition is formed from the conjunction of all of the class' categories. The OO-Browser does support Objective-C category browsing.

`Children`
: First level of classes below a class in the inheritance hierarchy. Those that directly inherit from a class.

`Class`
: A factory construct from which object instances are created. The OO-Browser displays classes along with their elements, categories and formal protocols.

`Class at Point`
: The class in a listing buffer whose name appears on the same line as point.

`Completion`
: The act of filling in the non-ambiguous part of a requested item, such as a class name or a file name, based on a list of possibilities.

`Declaration`
: A specification of a programmatic entity, for reference by other parts of a program. See also `Definition`. The declaration of a method specifies its signature but not its body.

`Default Class`
: A class that the OO-Browser automatically creates to categorize instances of constructs that are built-in to a language, such as class protocols or global functions. Default class names begin and end with square bracket delimiters, as in `[protocol]`.

`Definition`
: A complete, unambiguous description of a programmatic entity, For example, the interface and body of a method defines it.

`Decendants`
: All classes below a class in the inheritance hierarchy.

`Element`
: A feature or an instance of a class.

`Environment`
: A series of browser lookup tables and control variables that specify the set of classes and inter-class relationships with which the browser works.

`Environment File`
: A file used to store a browser Environment.

`Environment Specification`
: An unambiguous description of what to include in the construction of an Environment.

`Feature`
: A method, attribute, or other component of a class. Features may be public or private and in some languages, non-inheritable.

`Formal Protocol`
: See `Protocol`.

`Friend`
: In C++, a specially declared class or method which is granted access to the private parts of the class in which its friend declaration is found.

`Hyperbole`
: The flexible, programmable information management and viewing system documented by this manual. It utilizes a button-action model and supports hypertextual linkages. Hyperbole is all things to all people.

`Implementor`
: A class in which a particular element is defined. This does not include classes which inherit an element.

`InfoDock`
: InfoDock is an integrated productivity toolset for software engineers and knowledge workers. It is presently built atop XEmacs and is no longer maintained. An older version from 1999 may be found at infodock.sf.net. InfoDock has all the power of emacs, but with an easier to use and more comprehensive menu-based user interface. Most objections people raise to using emacs have already been addressed in InfoDock. InfoDock was meant for people who wanted a complete, pre-customized environment in one package.

`Initialization File`
: See `Personal Initialization File`.

`Instance`
: An object which has a particular class as its type. The class serves as a template for instance creation.

`Interface`
: See `Protocol`.

`Library Classes`
: Stable, seldomly changed classes that have been released for general usage.

`Listing Window`
: One of a number of browser windows used to display lists of entities. Inheritance relations are shown in listing windows via class name indentation.

`Lookup Table`
: A data structure used to speed response to user queries.

`Member` See `Feature`.

`Method` A callable function defined within one or more classes.

`Minibuffer Window`
The single line window at the bottom of an Emacs frame. It is used to interact with the user by displaying messages and prompting for input.

`Module` In Python, a namespace created by a code file used to group together global variables, functions and classes.

`Package` In Python, a namespace created by a code file used to group together global variables, functions and classes.

`Parents` The next level of classes above a specific class in the inheritance hierarchy. Those from which a class directly inherits.

`Point` The position within the current buffer that is immediately in front of the character over which the Emacs block cursor is positioned.

`Protocol` An interface specification to which a class conforms. Some languages use abstract classes for this purpose. Under Objective-C, Java and now Python, you may define formal protocols (also known as interfaces) which include a set of method signatures which a class must implement if it conforms to the protocol. One protocol may inherit from a list of other protocols, and thereby expand the set of methods which a conforming class must implement.

`Routine` See `Method`.

`Signature`
An interface specification for a method. It includes the method's class, type of return value and the types of its formal parameters.

`Smart Key` A context-sensitive key used within the OO-Browser and beyond. There are two Smart Keys, the Action Key and the Assist Key. The Action Key, bound to the shift-middle mouse key (or shift-left mouse key on a 2-button mouse), activates OO-Browser listing entries, jumps to identifier definitions in code and scrolls the current buffer line to the top of the window when pressed at the end of a line. (InfoDock users or those who set the variable, `hmouse-middle-flag`, to '`t`', may also use the middle mouse key as the Action Key).

The Assist Key does similar things to the Action Key when in the OO-Browser.

The {`C-h h d s`} Doc/SmartKeys menu item displays a full summary of Smart Key capabilities.

`System Classes`
Classes still in development whose interfaces are likely to change. They are typically part of a specific project and are often not meant to be reused elsewhere.

`Tag` A line from an OO-Browser internal lookup table that is used to match against the definition of a class element when browsing. Sometimes referred to as a signature tag. See also `Signature`.

`Top-Level Classes`
Classes without parents. Those at the top of the inheritance tree for a given Environment.

`Viewer Window`

The largest, bottom-most window in the browser used for displaying class source and help information.

Appendix B Menus

This appendix summarizes the commands available on the OO-Browser popup and menubar menus that mirror the keyboard-based commands discussed throughout the manual. See Appendix D [Commands], page 54, for a summary of the OO-Browser command names and the keys to which they are bound.

When running InfoDock, the `Mode-Menubars` button on the menubar will replace the menubar of global InfoDock commands with one specific to the OO-Browser. (Under XEmacs and GNU Emacs, a single `OO-Browser` pulldown menu automatically appears on the menubar when the point is within a listing window.)

B.1 OO-Browser Menu

The *OO-Browser menu* includes various help and browser exit commands. This menu appears as a top-level list of items within the listing window popup menu and the singular pulldown menu used under XEmacs and GNU Emacs. Its items are:

`About` Displays the OO-Browser version number and credits.

`Language-Manual`
: Displays the OO-Browser manual section of specifics for the language of the current Environment.

`Program-Manual`
: Displays the very beginning of the online version of the OO-Browser manual. The entire manual can be browsed in section order by simply using the {`SPC`} key. {`DEL`} moves backwards through the manual.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`What-is-New?`
: Displays a summary of user-visible changes in each OO-Browser release.

`---`

`Load-Env-by-Name`
: Menu which loads Environments based on user-given names.

`---`

`Copyright`
: Displays OO-Browser copyright information within the browser viewer window.

`Help-Commands`
: Displays a menu of OO-Browser commands within the browser viewer window. A press of the Action Key within any of the key bindings listed in this command menu invokes the associated command.

`Help-Mode`
: Summarizes the key bindings available within OO-Browser listing windows.

`Help-Mouse`
: Displays a summary of how the Action and Assist Keys behave across the contexts available throughout OO-Browser usage.

`---`

`Discuss-via-Email`
: Starts composing a new e-mail message to the OO-Browser discussion mail list.

`Get-Support-via-Email`
: Starts composing a new e-mail message to Deepware's support staff in which you can ask questions or make requests. This requires that you have pre-paid for support credits with Deepware.

`---`

`Reinitialize`
: Restores the OO-Browser user interface to its state upon startup.

`Exit-this-Listing`
: Returns to the previous OO-Browser listing buffer and discards the contents of the current window.

`---`

`Exit-Temporarily`
: Hides the OO-Browser's buffers and windows, for re-invocation later.

`Quit` Deletes all the OO-Browser listing buffers and all of its windows.

B.2 Class Menu

The *Class menu* contains operations that apply to individual class entries or a set of classes extracted from the Environment. Use the `List-Window menu` for the parallel set of commands that operate on all classes within a listing. Its items are:

`Concept-Manual`
: See Section 2.2 [Displaying Top-Level Classes], page 8.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`---`

`Edit-Definition`
: Edits within the viewer window the source definition of the current listing entry.

`View-Definition`
: Views the source definition of the current listing entry (the source is made read-only and point is left in the listing window).

`---`

`Edit-Named`
: Prompts for a class name and then edits the class definition within the viewer window.

`View-Named`
: Prompts for a class name and then views the class definition within the viewer window.

`---`

`Match-from-Listing`
: Prompts for a regular expression used to match against the names of all classes and features within the current Environment. The matches are displayed in a listing buffer which may then be filtered further with another match expression. {`RET`} ends the matching.

`Where-is-Named?`
: Prompts for a class or feature name and then displays in the viewer window the full path of the source file where the name is defined.

`Where-is-Entry?`
: Displays in the viewer window the full path of the source file which defines the present listing entry.

`---`

`Ancestors`
: Generates an ancestor tree listing for the current class. The ancestors branch down and outward from the current class. See Section 2.6 [Browsing Descendants and Ancestors], page 9, for information on how to invert the listing.

`Attributes`
: Lists the attributes of the current class. If `br-inherited-features-flag` is '`t`', all attributes including inherited attributes of the class are shown; otherwise, only the attributes lexically defined within the class are shown.

`Children` Lists the children of the current class.

`Descendants`
: Generates the descendency tree for the current class.

`Features` Lists the features of the current class. If `br-inherited-features-flag` is '`t`', all features including inherited features of the class are shown; otherwise, only the features lexically defined within the class are shown.

`Implementors`
: Displays a list of classes which contain definitions for the current <protocol/interface> entry.

`Info` Displays language-specific summary information for the current class entry.

`Level` Shows the current class location within its inheritance graph, that is, among its ancestors and descendants.

`Parents` Lists the parents of the current class.

`Protocols`
: Displays the protocols to which the current class or protocol conforms, including inherited ones.

`Routines` Lists the routines (methods) of the current class. If `br-inherited-features-flag` is '`t`', all routines including inherited routines of the class are shown; otherwise, only the routines lexically defined within the class are shown.

`Class-Statistics`
: Displays within the minibuffer a statistics summary for the current class. This generally shows the number of parents and children that the class has.

B.3 Environment Menu

The commands on the *Environment menu* manipulate entire Environments. Its items are:

`Concept-Manual`
: See Chapter 1 [Working with Environments], page 4.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`Create-or-Load`
: Invokes the OO-Browser on a new or existing Environment.

`Display-Env-List`
: List Environment names and associated Environment files.

`Rebuild` Rescans the System and Library code directories associated with the current Environment to update the entire Environment.

`Statistics`
: Displays a summary for the current Environment in the viewer window.

`Add-Name` Associates a new name with an existing Environment.

`Change-Name`
: Renames an Environment.

`Remove-Name`
: Deletes an Environment name but not the Environment itself.

`Replace-Env-of-Name`
: Changes which Environment is associated with a particular name.

`Delete-Class`
: Deletes a class from the current Environment.

`Save` Prompts for a file to which to save the current Environment, with a default of the current Environment file name.

B.4 Feature Menu

The *Feature menu* supplies commands that apply to feature (attribute and method) entries. (More precisely, these commands work with any kind of element entry.) Its items are:

`Concept-Manual`
: See Section 2.8 [Browsing Elements], page 11.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`---`

`Edit-Definition`
: Edits within the viewer window the source definition of the current listing entry.

`View-Definition`
: Views the source definition of the current listing entry (the source is made read-only and point is left in the listing window).

`---`

`Edit-Declaration`
: Edits within the viewer window the declaration (signature) of the current listing entry.

`View-Declaration`
: Views the declaration (signature) of the current listing entry.

`---`

`View-Friend-Def`
: With point on a friend listing entry, views its source code definition.

`---`

`Edit-Named`
: Prompts for a feature name and then edits the feature definition within the viewer window.

`View-Named`
: Prompts for a feature name and then views the feature definition.

`---`

`Current-Attributes`
: See `Class/Attributes`.

`Current-Features`
: See `Class/Features`.

`Current-Routines`
: See `Class/Routines`.

`---`

`All-Attributes`
: See `List-Window/All-Attributes`.

`All-Features`
: See `List-Window/All-Features`.

`All-Routines`
: See `List-Window/All-Routines`.

`---`

`Implementors`
: Displays a list of classes which contain definitions for the current element name.

`Signature`
: Shows within the viewer window the full feature signature for the current listing entry.

B.5 Graphical Menu

The *Graphical menu* creates and deletes graphical, tree-oriented views of Environment information related to listing entries. Its items are:

`Concept-Manual`
: See Section 2.22 [Browsing Graphically], page 18.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`---`

`Class-Descendants-View`
: Displays within a graphical viewer the descendency tree for the current class. If `Options/Graphical-Descendant-Features` is enabled, the features of each descendant are added to the view.

`Listing-Descendants-View`
: Displays within a graphical viewer the descendency trees for all listing entries within the current buffer. If `Options/Graphical-Descendant-Features` is enabled, the features of each descendant are added to the view.

`Listing-Graphical-View`
: Displays within a graphical viewer the tree of listing entries from the current buffer. Each entry node in this view may be selected with the mouse and its corresponding source will be displayed within the editor for either viewing or editing depending on which mouse key was used.

`---`

`Kill-Graphical-Views`
: Deletes all existing OO-Browser graphical views.

B.6 List-Window Menu

The *List-Window menu* offers commands that operate on all of the entries within the current listing window at once. Its items are:

`Concept-Manual`
: See Chapter 2 [Using the OO-Browser], page 8.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`---`

`Write (Save as)`
: Prompts for the name of a new or existing file and writes the narrowed portion of the current listing buffer to the file. Any existing contents of the file are overwritten.

`---`

`Count-Entries`
: Counts the number of entries visible in the current listing buffer and displays the result within the minibuffer window.

`Order-Entries`
: Alphabetizes the current listing window entries. Ignores any leading whitespace and removes any duplicates found.

`---`

`All-Ancestors`
: Displays ancestor trees for all entries within the current listing window which have ancestors.

`All-Attributes`
: Displays attributes for all entries within the current listing window which have attributes.

`All-Children`
: Displays the children for all entries within the current listing window which have children.

`All-Descendants`
: Displays descendant trees for all entries within the current listing window which have descendants.

`All-Features`
: Displays features for all entries within the current listing window which have features.

`All-Implementors`
: For each element in the current listing, displays a list of classes which contain definitions for it. Ignores classes which inherit such definitions.

`All-Levels`
: Shows for each class within the current listing window its inheritance graph, i.e. where it is located among its ancestors and descendants.

`All-Parents`
: Displays the parents for all entries within the current listing window which have parents.

`All-Protocols`
: Display the protocols to which each class and protocol within the current listing window conforms, including inherited ones.

`All-Routines`
: Displays routines (methods) for all entries within the current listing window which have routines.

`---`

`Show-All-Classes`
: Displays the list of all classes.

`Show-All-Lib-Classes`
: Displays the list of all Library classes.

`Show-All-Sys-Classes`
: Displays the list of all System classes.

`Show-Top-Classes`
: Displays the list of all top-level classes, i.e. those without parents.

`Show-Top-Lib-Classes`
: Displays the list of all top-level Library classes.

`Show-Top-Sys-Classes`
: Displays the list of all top-level System classes.

`---`

`Narrow-by-10`
: Narrows the OO-Browser listing windows by 10 characters.

`Widen-by-10`
: Widens the OO-Browser listing windows by 10 characters.

`---`

`Exit-this-Listing`
: Returns to the previous OO-Browser listing buffer and discards the contents of the current window.

B.7 Options Menu

The *Options menu* toggles major OO-Browser option settings. Its items are:

`Concept-Manual`
: See Chapter 3 [OO-Browser Options], page 23.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`---`

`Keep-Viewed-Classes`
: Toggles whether or not multiple class buffers are left around after viewing. If off (the default), each viewed class buffer is deleted after use. Classes displayed for

editing are not affected; they are always left around for further editing unless explicitly deleted.

`Graphical-Descendant-Features`
: Toggles whether or not features are added to any class listing displayed in graphical form. The default is not to add features to graphical displays.

`List-Protocols-with-Classes`
: Toggles whether or not protocols (interfaces) are included in listings of all classes or top-level classes. The default is to include them under languages that support protocols.

`Show-Inherited-Features`
: Toggles whether or not feature listings include inherited features. The default is to include them. If off, only those features lexically included within a class are shown.

`Use-Vi-as-Editor`
: Toggles whether or not the OO-Browser sends source code to Vi for editing and viewing, instead of to the viewer window.

  #cindex 2-button mouse #cindex 3-button mouse

`3-Button-Mouse`
: Toggles which mouse key is used for displaying items from listing buffers. With this enabled, the middle mouse buttons is used for this purpose. Otherwise, the left mouse button is used.

B.8 View-Window Menu

The *View-Window menu* is used to manipulate the viewer window display from within a listing window. Its items are:

`Concept-Manual`
: See Section 2.7 [Viewing and Editing], page 10.

`Menu-Manual`
: Displays this section of the OO-Browser manual.

`---`

`Select-Code-Buffer`
: Displays within the viewer window a list of all existing buffers of source code for the current OO-Browser language. A press of the Action Key, {`SPC`}, or {`q`} on an entry line displays the associated buffer.

`---`

`Full-Frame`
: Deletes all browser windows except for the viewer window. This is used to get a full-frame view of a source file.

`Kill-Buffer`
: Kills the current buffer within the viewer window and redisplays the initial OO-Browser command help buffer.

`Move-To-or-From`
: Moves point back and forth between the viewer window and the last recorded browser listing window.

`Scroll-Backward`
: Scrolls the viewer backward one windowful.

`Scroll-Forward`
: Scrolls the viewer forward one windowful.

`Scroll-Backward-One-Line`
: Scrolls the viewer window backward one line.

`Scroll-Forward-One-Line`
: Scrolls the viewer window forward one line.

`To-Buffer-Beginning`
: Scrolls to the beginning of the viewer buffer.

`To-Buffer-End`
: Scrolls to the end of the viewer buffer.

Appendix C Features

- Support for C, C++, Common Lisp and its Object System (CLOS), Eiffel, Java, Objective-C, Python and Smalltalk class browsing is included. Additionally, support for browsing large amounts of material in Info format by node name (a popular online documentation format with cross references and hierarchical structure) is included. All languages provide class browsing via either a textual or a graphical interface.
- Method and typically attribute browsing is supported for all languages except Smalltalk. CLOS supports browsing all elements defined with `(def` constructs. In-source feature browsing is also supported for all of these languages. One simply selects a feature name to jump to its corresponding source. Method name overloading in C++ and inherited feature renaming in Eiffel are fully supported.
- Under C++, one can click on a method call, function call or attribute reference to jump to its associated definition. If multiple definitions are possible, a structured dynamic list of possible method signatures are shown and can be clicked upon to jump to any selected definition.
- Under C++, friend classes and functions may be browsed easily.
- C code browsing is supported for C++, Objective-C and C source code.
- Objective-C category and formal protocol browsing is supported.
- C++ parameterized template classes and methods are supported.
- Java abstract and native (externally defined) method browsing is supported.
- All classes that implement a particular feature name, protocol (or interface) name, or class category name may be listed and then browsed.
- Immediate switching among languages is allowed. One can switch from Eiffel browsing to C++ browsing in an instant, if so desired. Or simply run two OO-Browsers side by side (in separate editor sessions)..
- Multiple inheritance support is built-in, where applicable.
- Statistics on classes and Environments may be displayed.
- Language-specific class information may be shown. Presently this feature is supported in a minor way under Python and more extensively under Eiffel, where a listing of class parents, attributes, routines and best guess (highly accurate) list of routine calls may be displayed. Outputs from the Eiffel `short` and `flat` commands may also be shown.
- Library (stable) and System (in development) classes may be maintained and listed separately or together. Any number of Libraries and Systems may be combined for listing in a single Environment. There are no fixed limits on the number of classes per Environment nor on the number of Environments that may be browsed.
- All source code is included and is heavily documented.
- Machine-independent mouse support is included along with an extremely intuitive point and click interface that uses just two mouse keys. The OO-Browser is pre-configured for use with the X window system and Microsoft Windows under InfoDock, GNU Emacs and XEmacs. Online mouse usage help is always one key away.
- Popup and pulldown command menus are available under InfoDock, GNU Emacs and XEmacs.

- The OO-Browser help display gives short descriptions of all of the commands and key bindings available in the browser. By clicking on any such selection, the corresponding command is executed.
- One may also click on class names to see ancestors, descendants or the class itself. Just select a class name and the OO-Browser immediately will display or edit the class source. Once a class file has been loaded, one can quickly switch to it by selection from a menu of such files.
- For a number of languages, one may also select a feature (method) name or declaration and move directly to the definition of the feature. The browser accounts for the complexities of member name overloading in C++ and unlimited feature renaming in Eiffel so that you need not. Just click on a declaration and watch the browser display jump to the proper definition.
- In C++, one can jump to the declaration of a listing entry or be prompted within any buffer for a class and feature name whose declaration one wants to browse. One can jump back and forth between declarations and their associated definitions (between header and code files) with a single command.
- Jump back to a previously visited class or feature by selecting from a list of recently visited buffers.
- OO-Browser commands may also be invoked from the keyboard, allowing unrestricted use via standard terminal interfaces.
- Building Environments is fast compared to many other tools. Browser startup, once an Environment has been built, is very fast. Response times on workstations are excellent; for example, in one test case years ago, less than two real seconds were required to display a set of complex inheritance graphs involving over 400 classes.
- An X-specific or Windows-specific hierarchy display browser is included. It provides views of class inheritance structure and lexically included elements, which allows for quick random access to entire Environments. A click on a class or element name immediately jumps to it in the editor, providing rapid, visual browsing. One can pop up several graphical browsers to gain different views of classes in the same or in multiple environments. All graphical browsers can communicate with a single textual browser, so one can quickly display and edit classes from different environments (even different languages). Multiple inheritance is handled through repetition of nodes throughout the tree; repeated nodes are followed by ellipses to indicate multiple inheritance.
- The OO-Browser uses class source code only, hence no compiler is necessary for proper browser operation. This allows one to explore class libraries without the need for additional tools.
- Class inheritance networks may be displayed. Either a single inheritance level (parents or children) or the entire inheritance network (ancestors or descendants) for a set of classes may be shown.
- Class files may be added as a group by specifying a root directory below which all class files are found, including those in subdirectories.
- A menu of class files can be displayed for random access to specific code modules.
- On startup, the OO-Browser lists all currently known classes within a particular Environment. Any desired classes may be found by searching or by matching a regular

expression or string to the set of class names. This may be done repeatedly to achieve an "and"-type relational query effect.

- The number of listing windows is limited only by the frame width and the width setting used for listing windows.
- The OO-Browser is adaptable to any class-based object-oriented language.
- The OO-Browser works with the powerful, freely distributable, GNU Emacs editor; it works on any UNIX system display supported by Emacs. It is included as part of InfoDock, the integrated development environment, and is also compatible with XEmacs. Alternative editors may also be used to view or to edit source code displayed by the browser.
- All OO-Browser outputs are text which may be edited as desired or saved to files.
- OO-Browser functions may be used standalone within the editor without utilizing the multi-windowed browser interface. One useful example is to point to a class name such as a parent class in the text of another class and have the parent's source appear in an editable fashion.
- The user need not know the location of class source; the browser will display or edit a class based solely upon its class name.
- A single key provides ascending or descending ASCII ordering of class names, including those from inheritance trees. Classes may be easily located by matching a regular expression or string to the set of class names in an Environment, with repeated searches incrementally narrowing the selected set.
- The browser is tailorable to any class-based object-oriented language. It works best with languages that focus on static class creation such as Eiffel and C++.
- The OO-Browser is built to integrate with the powerful GNU Emacs and XEmacs editors and the even more powerful InfoDock environment; it works on any UNIX, DOS, Windows or Macintosh system display supported by Emacs. Most browser commands may be executed by direct selection, providing a very natural interface.

Appendix D Commands

The following documentation is meant for programmers who want to modify the OO-Browser. It is included here since some users of the OO-Browser may find it useful. All commands that are bound to keys and that are specific to the OO-Browser are listed here, plus a few other commands. Within each command description, identifiers shown in all capitals are the names of the command's formal arguments; all formal arguments are presented in the order in which they are required by the command. If a command takes optional arguments, the first optional argument is labeled *optional*; all following arguments are assumed to be optional.

`br-ancestors {a}`
: Display ancestor tree whose root is the current class. With optional prefix ARG, display all ancestor trees whose roots are in the current listing. With no ARG or if ARG = -1 or `br-invert-ancestors` is '`t`', the current class ancestry tree is inverted. That is, it shows branches going down towards the root class, so that parents appear above children. If ARG < -1 or `br-invert-ancestors` is '`t`' and ARG > 1, then the ancestry trees of all classes in the current listing are inverted.

`br-at {@}`
: Display the current class location in the inheritance graph. The class is displayed among both its ancestors and descendants. With optional prefix ARG, display the locations for all classes in the current listing.

`br-attributes`
: Display attributes of the current class (prefix ARG = 1) or of the current listing if ARG is other than 0 or 1.

  With ARG = 0, the value of the variable, `br-inherited-features-flag`, is toggled and no other action is taken.

  If `br-inherited-features-flag` is '`t`', all attributes of each class are shown. If '`nil`', only lexically included attributes are shown and if the attributes of a single class are requested and none are defined, the class definition is displayed so that its attribute declarations may be browsed.

`br-buffer-menu {b}`
: Display in the viewer window a selection list of buffers for the current browser language.

`br-categories {C}`
: Display categories directly associated with the current class. This does not include any categories which the class inherits. With optional prefix ARG, display categories of all classes in the current listing.

`br-children {c}`
: Display the children of the current class. With optional prefix ARG, display the children of all the classes in the current listing.

`br-class-stats {M-c}`
: Display a statistics summary for the current class. Optional prefix arg PROMPT means prompt for a class name.

`br-copyright`
: Display the OO-Browser copyright information in the viewer window.

`br-count {#}`
: Count the number of entries visible in current listing buffer. Print the text result in the minibuffer when called interactively.

`br-delete {C-c C-d}`
: Delete a class from the current Environment. Does not alter descendency relations. Optional prefix arg PROMPT means prompt for the class name.

`br-descendants {d}`
: Display the descendant tree whose root is the current class. With optional prefix ARG, display all descendant trees whose roots are the classes in the current listing.

`br-edit-entry {e}`
: Edit the source code for any browser listing entry, such as a class or a feature. Optional prefix arg PROMPT means prompt for the entry name; automatically prompt if called interactively outside of a listing window, e.g. within a source code buffer when the browser user interface is not displayed.

`br-entry-info {i}`
: Display attributes of the current entry in the viewer window.

`br-env-load {C-c C-l}`
: Load an OO-Browser Environment or specification from optional ENV-FILE, ENV-NAME or `br-env-file`. Non-nil PROMPT means prompt the user before building the Environment. Non-nil NO-BUILD means skip the build of the Environment entirely. Return '`t`' if the load is successful, else '`nil`'.

`br-env-rebuild {C-c C-e}`
: Rescan System and Library sources associated with the current Environment. When given a prefix arg, DEBUG-FLAG, it will output a debugging backtrace if any error occurs during scanning (InfoDock and XEmacs only).

`br-env-save {C-c C-s}`
: Save the modified Environment to a file given by optional SAVE-FILE or `br-env-file`.

`br-env-stats {M-e}`
: Display a summary for the current Environment in the viewer window. With optional prefix ARG, display class totals in the minibuffer.

`br-exit-level {x}`
: Return to prefix ARGth previous inheritance level listing. The command is ignored with ARG < 1.

`br-feature-edit-declaration {C-c M-j}`
: Prompt with completion for a CLASS::FEATURE argument and then edit the associated declaration. If point is on a feature definition signature in a code buffer (prior to any of its arguments), the default is to edit that feature's declaration. An error is signaled if the declaration is not found. Presently, this command works in C++ buffers exclusively.

`br-feature-signature {F}`
: Show the full feature signature in the view window. With optional prefix ARG, display signatures of all features from the current listing buffer.

`br-features {f}`
: Display features/elements of the current class (prefix ARG = 1) or of the current listing if ARG is other than 0 or 1.

  With ARG = 0, the value of the variable, `br-inherited-features-flag`, is toggled and no other action is taken.

  If `br-inherited-features-flag` is '`t`', all features of each class are shown. If '`nil`', only lexically included features are shown and if the features of a single class are requested and none are defined, the class definition is displayed so that its feature declarations may be browsed.

`br-find {C-c M-f}`
: Prompt with completion for a class or element name from the current Environment and display its definition for editing. (This command is available within source code buffers.)

`br-help {h} or {?}`
: Display OO-Browser operation help information in the viewer window.

`br-help-ms {H}`
: Display OO-browser mouse usage help information in the viewer window.

`br-implementors {I}`
: Display a list of classes which contain definitions for the current element name. Ignore classes which inherit such definitions. With optional prefix ARG, display implementors of all elements within the current listing.

`br-kill {C-c C-k}`
: Kill the buffer in the viewer window and redisplay help text.

`br-lib-rebuild {L}`
: Rescan Library components of the current Environment.

`br-lib-top-classes {l}`
: Display a list of the top-level Library classes. With prefix ARG, display all Library classes.

`br-match {m}`
: Show all class names in the current Environment that contain optional EXPR. A '`nil`' value of EXPR means prompt for a value. With optional prefix ARG, EXPR is treated as a string. By default, it is treated as a regular expression. AGAIN non-nil shows the number of classes MATCHED from the last search, allowing repeated narrowing of the search set. An empty EXPR when AGAIN is '`nil`' matches to all classes in the Environment.

`br-match-entries {M}`
: Show all entries in the current listing that contain optional EXPR. A '`nil`' value of EXPR means prompt for a value. With optional prefix ARG, EXPR is treated as a string. By default, it is treated as a regular expression. AGAIN

non-nil means show the number of entries MATCHED from the last search, allowing repeated narrowing of the search set. An empty EXPR when AGAIN is '`nil`' matches to all entries in the listing.

`br-next-entry {C-n}`
Move point vertically down prefix ARG number of lines in a listing buffer.

`br-order {o}`
Order current browser listing window entries. With prefix ARG other than 1 (the default), don't remove leading space from entry lines before ordering. Negative ARG means order in descending Ascii sequence, otherwise order in ascending sequence.

`br-parents {p}`
Display the parents of the current class. With optional prefix ARG, display parents of all the classes in the current listing.

`br-prev-entry {C-p}`
Move point vertically up prefix ARG number of lines in a listing buffer.

`br-protocols {P}`
Display the protocols to which the current class or protocol conforms, including inherited ones. With optional prefix ARG, display protocols of all classes and protocols in the current listing.

`br-quit {q}`
Quit the OO-Browser. With optional prefix ARG, delete window configurations and listing buffers associated with the browser.

`br-refresh {C-c C-r}`
Restore the OO-Browser to its state upon startup.

`br-report-bug {C-c C-b}`
Report a bug or send some other kind of message to the OO-Browser maintainers.

`br-resize-narrow {C-x -}`
Narrow listing windows by 10 characters.

`br-resize-widen {C-x +}`
Widen listing windows by 10 characters.

`br-routines`
Display routines of the current class (prefix ARG = 1) or of the current listing if ARG is other than 0 or 1.

With ARG = 0, the value of the variable, `br-inherited-features-flag`, is toggled and no other action is taken.

If `br-inherited-features-flag` is '`t`', all routines of each class are shown. If '`nil`', only lexically included routines are shown and if the routines of a single class are requested and none are defined, the class definition is displayed so that its routine declarations may be browsed.

`br-sys-rebuild {S}`
Rescan System components of the current Environment.

`br-sys-top-classes {s}`
: Display a list of top-level System classes. With prefix ARG, display all System classes.

`br-to-from-viewer {C-c C-v}`
: Move point to the viewer window or back to the last recorded listing window.

`br-toggle-keep-viewed`
: Toggle the value of the `br-keep-viewed-classes` flag.

`br-show-all-classes {A}`
: Display a list of all Environment classes.

`br-show-top-classes {t} or {T}`
: Display a list of top-level classes. With prefix ARG, display all Environment classes.

`br-tree {M-d}`
: Start the appropriate tree application with a descendency tree of the current class. With optional prefix ARG, include a descendency tree for each class in the current listing buffer.

`br-tree-features-toggle {M-f}`
: Toggle between showing and hiding features when `br-tree` is invoked to display descendants graphically.

`br-tree-graph {M-g}`
: Start the appropriate tree application with the tree from the current listing buffer.

`br-tree-kill {M-k}`
: Kill all current `xoobr` sub-processes.

`br-unique {u}`
: Eliminate adjacent duplicate entry names from the current listing window. If two adjacent entries look the same, one is eliminated, even if they refer to different class elements.

`br-version {C-c #}`
: Display the OO-Browser version number and credits.

`br-view-entry {v}`
: Display source for any browser listing entry. Optional prefix arg PROMPT means prompt for an entry name; automatically prompt if called interactively outside of a listing window, e.g. within a source code buffer when the browser user interface is not displayed.

`br-view-friend {V}`
: With point on a friend listing entry, view its source code definition. With optional OTHER-WIN non-nil, display in another window. With optional SIG-AT-POINT-FLAG non-nil, assume point is within a friend signature in a source buffer. (C++ only).

`br-view-full-frame {1}`
: Delete all windows in the selected frame except for the viewer window.

`br-viewer-beginning-of-buffer {<}`
: Scroll to the beginning of the viewer window buffer from within a listing window.

`br-viewer-end-of-buffer {>}`
: Scroll to the end of the viewer window buffer from within a listing window.

`br-viewer-scroll-down {DEL}`
: Scroll the viewer window downward ARG lines or a windowful if no ARG.

`br-viewer-scroll-down-by-line {,}`
: Scroll the viewer window from within a listing window to show prefix ARG more prior lines (default is 1).

`br-viewer-scroll-up {SPC}`
: Scroll the viewer window upward ARG lines or a windowful if no ARG.

`br-viewer-scroll-up-by-line {.}`
: Scroll the viewer window from within a listing window to show prefix ARG more following lines (default is 1).

`br-where {w} (in a listing window) {C-c M-w} (elsewhere)`
: Display in the viewer window and return the full path of the defining file for a browser listing entry. Optional prefix arg PROMPT means prompt for the entry name; automatically prompts if called interactively outside of a listing window, e.g. within a source code buffer when the browser user interface is not displayed.

`br-write-buffer {C-c C-w}`
: Write the narrowed portion of the current browser buffer to a file.

Appendix E References

[**Meyer 88**]
Meyer, Bertrand. Object-oriented Software Construction. Prentice Hall International: UK, 1997.

[**Meyer 89**]
Meyer, Bertrand. Eiffel: The Language. Interactive Software Engineering: Santa Barbara, CA, 1989. (Also published by Prentice Hall.)

[**Goldberg 83**]
Goldberg, Adele and David Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[**Stallman 93**]
Stallman, Richard. *GNU Emacs Manual.* Free Software Foundation: Cambridge, MA, 1993.

[**Java 95**] *The Java Language Specification.* Sun Microsystems Computer Corporation: Mountain View, CA, February 1, 1995.

Key Binding Index

#

,

.

<

=

>

1

A

C

D

E

F

H

I

J

L

M

O

P

Q

R

S

T

V

W

X

Command and Variable Index

B

C

Concept Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

X