Bluefish: The Definitive Guide

User's, Developer's, and Documentarian's Guide for Bluefish version 1.0

Daniel Blair
Michèle Garoche
Anita Lewis
Alastair Porter
Denny Reeh
Olivier Sessink
Scott White

94-98 pages

Bluefish: The Definitive Guide: User's, Developer's, and Documentarian's Guide for Bluefish version 1.0

by Daniel Blair, Michèle Garoche, Anita Lewis, Alastair Porter, Denny Reeh, Olivier Sessink, and Scott White
Logo art: Dave Lyon

Table of Contents

List of Figures

List of Examples

List of Procedures

Preface

About this Manual

Bluefish has a large feature set, allowing the user to customize the editing experience in numerous ways. This manual targets both novice and advanced users, providing a full resource for everyone.

Chapters 1, 2, and 3 are highly recommended for anyone new to Bluefish. They present general information, installation instructions, and an introduction to the main features of Bluefish.

Chapter 4 explains how to debug Bluefish.

Chapter 5 contains a nearly complete feature reference, useful for advanced users interested in customizing Bluefish.

Chapter 6 provides guidance for developers, including code formatting styles and a reference for all the source files.

The manual targets the *end user*. To that end, we have tried to use a simple, well-explained approach whenever possible. Some typographic conventions are denoted below:

- Any URLs are denoted like this: http://bluefish.openoffice.nl
- Shortcuts look like this: **Ctrl-S**
- Menu options are displayed like this: File. However, many of Bluefish's menus are quite complex. When referring to submenus, options are separated by an arrow, like: File->Open (**Ctrl-O**). The default keyboard shortcut is shown in parenthesis.
- When referring to user input, like issuing commands to the command prompt, a monotype font is used:

```
$ foo -bar | bang -l
```

Note

Do not write the $ character - it simply identifies the command prompt. For commands requiring root access, the prompt is shown as a #.

Finally, if you find errors in this manual or wish to write new sections, join the mailing list and let us know. Guidelines for this manual can be found in Appendix C, *Guidelines for Writing this Manual*.

What is Bluefish?

Bluefish is a powerful editor for experienced web designers and programmers based on the GTK2 GUI interface. Bluefish supports many programming and markup languages, but focuses on editing dynamic and interactive websites.

Bluefish is not a WYSIWYG[1] text editor. This is deliberate, allowing the programmer to stay in full control. To facilitate the editing process, a large number of features are at your disposal. For inserting markup and code, there are tool bars, dialogs, and predefined/user-customized menus. Syntax highlighting, advanced search/replace functionality, scalability and language function references make Bluefish a powerful tool for development.

History of Bluefish

Bluefish development started under a different name. A good and free text editor targeted towards web development was not available. Olivier Sessink started the project ProSite. Chris Mazuc also started an HTML editor. On a GTK development mailing list, Olivier Sessink and Chris Mazuc saw each others postings, and decided to team up. Olivier had a basic editor, Chris had many HTML dialogs ready. After merging the code this was for a while known as the Thtml editor.

After a while Neil Millar joined the project to add weblint integration and a color dialog. Because the project became larger and more mature, a logo was wanted. After many discussions about boring logos, Neil Millar came up with a cute blue fish. Because this logo was appreciated by all, the name changed into the final name Bluefish.

After this initial stage, many developers, translators, testers and users joined the project.

Several years have passed since the first Bluefish release. Since that time, the fish has gained a reputation as an excellent editor, with qualities like stability, usability and numerous features. Also, Bluefish is small, fast and efficient, making it usable even on slow machines.

[1]What You See Is What You Get

Main Features of Bluefish

This list will give you an overview of the most important or outstanding features found in Bluefish:

- A What You Write Is What You Get interface
- Multiple document interface, will easily open 500+ documents (tested 3500 with documents simultaneously).
- Customizable syntax highlighting based on Perl compatible regular expressions, with subpattern support. Default patterns are included for:
  - C
  - cfml
  - ColdFusion
  - Gettext po
  - HTML
  - Java
  - Pascal
  - Perl
  - PHP
  - Python
  - R
  - XML
- Anti-aliased text window
- Multiple encodings support, can convert between different character sets, supports multibyte characters, Unicode, UTF8, etc.
- Nice wizards for startup, tables, frames, and others
- Dialogs for many HTML tags, with all their attributes
- HTML tool bar and tear-off menus
- User-customizable tool bar for quick access to often used functions
- Open files based on filename patterns and/or content
- Fully featured image insert dialog
- Thumbnail creation and automatically linking of the thumbnail with the original image
- Multi-thumbnail generation for easy creation of photo albums or screen shot pages
- Line numbers along the document
- Bookmarks for lines across multiple documents, with bookmark browser
- A custom menu, specify your own tags or sets of code, and define your own dialogs
- Custom search and replace pattern support for the Custom menu
- Very powerful search and replace, allowing POSIX and Perl Compatible regular expressions and sub-pattern replacing
- Excellent undo/redo functionality
- Configurable recent documents and recent directories functionality
- Spell checking
- Translations in more than *fourteen languages*
- User customizable integration with many programs, including weblint, tidy, make, javac, etc.
- XML based function reference. Currently, references are included for HTML and PHP. A GTK reference is available, and support for Perl and Python will be added. You may also create your own function reference. The XML format is described later in the manual.

As Bluefish is a part of a larger desktop environment, we have focused on making the GUI consistent with the Gnome HIG[2]. However, we prefer not following it in every detail, as some parts are intended *for the end user*, while Bluefish is *for the programmer*.

Some features from v0.7 (GTK1) are not yet implemented. The main missing piece is project management, which will be implemented before v1.0. If you depend on this feature, v0.7 may still be the version of choice.

How Stable is Bluefish?

Quite stable! The Bluefish developers aim to produce code that neither crashes nor leaks memory. Of course, that is not always easy to do. Leaks and crashes are often fixed in CVS as soon as they are discovered and hunted down. In addition to Bluefish's large user base, the developers use Bluefish for their daily work. So, fixing bugs and preventing crashes is always a major priority. However, some nags still exist. One example being the issue of slightly sluggish copy/paste functions.

For an updated list of open bugs, please go to the http://bfwiki.tellefsen.net/?pagename=ToDoList page on the Bluefish WiKi.

We appreciate any and all contributions! Please tell us if Bluefish crashes on you :-).

[2]GNOME Human Interface Guidelines, accessible at http://developer.gnome.org/projects/gup/hig/

Contact Us

We, the Bluefish development team, welcome all comments, user requests, constructive criticisms, and contributions. Are you curious or seeking information regarding Bluefish? Would you like to contribute by translating Bluefish or its manual? Here are your options:

- http://bluefish.openoffice.nl/ - The main website where you will find news, updates and more information.
- http://bfwiki.tellefsen.net/ - The Bluefish WiKi is the notebook for the developers, containing a lot of information. This includes, but is not limited to: updated project road maps, status of translations, feature requests, and open bugs.
- You can subscribe to the Bluefish mailing list by sending an email containing “subscribe bluefish-dev” to `<bluefish-dev-request@lists.ems.ru>`.
- Do you want to help translate Bluefish? Please let us know by dropping an email to Walter Echarri `<wecharri(at)arnet.com.ar>`, our friendly translation maintainer.
- If you have a general question, drop an email to `<bluefish(at)bluefish.openoffice.nl>`.

Chapter 1. Getting Bluefish

Choosing a Version

Currently, four versions of bluefish are available:

- The GTK1-version (v0.7) is deprecated and no longer updated, but is the choice for those of you still running GTK1.
- The latest GTK2-version (v1.0) is the version of choice for most users, and is regarded as stable enough for daily use.
- The latest development snapshot is always one step further than the latest stable release. In it, you will find some new features, bug fixes, and a prettier GUI. The catch is that it may have unfinished or buggy features. Try this if you want to see new features or are bothered by a bug in the latest stable release.
- CVS is the bleeding edge of Bluefish development. You may find that the CVS version has several bug fixes and enhancements, however, it may also contain new, inadvertent bugs. You will also need the CVS version if you want to contribute a patch. Although the CVS version may be unusable for short periods of time, it is often stable enough for daily use.

As commented in the section called “How and When Updates are Released”, the long time between stable releases makes the CVS snapshots and current CVS an enticing choice.

If you want the latest and greatest, read the section called “Latest Developmental Version” below. If you simply want to use Bluefish, read the section called “Latest Stable Version” for how to get the latest stable package for your system.

How and When Updates are Released

Due to the small number of volunteer developers, the progression of Bluefish's development often fluctuates. For this reason, a long time may pass between each release. After all, the developers volunteer their time and effort because they actually want to *use* Bluefish :-)

Because of the long periods of time between releases, the current CVS or CVS snapshots may be what you want to use. Bugs will be fixed and new features introduced. We do try to keep the CVS version usable at any time (actually, the CVS version is used by most of the development team on a daily basis).

Operating Systems Supported by Bluefish

Bluefish has been reported to work on a number of systems. The Bluefish team mainly support these platforms:

- Mandrake Linux
- Red Hat Linux
- Fedora Core
- Debian Linux
- FreeBSD

Actually, any GNU/Linux distribution with GTK2 is fine and many distributions include Bluefish. In fact, Bluefish will likely work quite well on any POSIX compatible OS where GTK2 is available. Bluefish has been reported to work on the following:

- NetBSD - distributed in pkgsrc
- OpenBSD - available through their ports system
- SGI IRIX - see http://freeware.sgi.com/
- Mac OS X
- Sun Solaris
- Tru64
- AIX
- HP-UX
- Win32-cygwin - with a few nags.

Latest Stable Version

Many Linux distributions ship a version of Bluefish or make it available through their package systems. For example, Bluefish is available through the Debian apt-system and FreeBSD's ports. You may check if Bluefish is available through your favorite software installer.

However, the main source is the Bluefish website, where the software and a few contributions are available. The download page is reachable at http://bluefish.openoffice.nl/download.html. Here, you may download the source code and binary packages for Debian, Red Hat/Fedora, and Mandrake.

Latest Developmental Version

To get the latest version of Bluefish you will need to download the source files from our CVS repository.

CVS[1], a version control system, is a widely used software development tool. It keeps track of changes to the source code, and allows for reversion to previous states. If you want to read more about CVS, have a look at the CVS-book by Karl Fogel, available at http://cvsbook.red-bean.com/cvsbook.html.

The Bluefish project's CVS repository is generously hosted by SourceForge.net[2]. For more information about them, see their site. The project homepage is http://sourceforge.net/projects/bluefish. Our CVS repository contains the current Bluefish source code, including this manual. The repository is accessible by anyone, and is updated almost daily by the developers.

To access the repository, you need a few small utilities. They are likely to be available through your favorite source of software (ports, apt, etc). The above-mentioned CVS book is a great source for information.

Here's how you get the source.

Procedure 1.1. Getting the source

1. The first step is to cd to the directory in which you want to put the sources.
2. Next, log in using the command:

   ```
   $ cvs -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/bluefish \
                                   login
   ```

   Note
   Hit Enter at the password prompt.
3. The next step is to *check out* the CVS module containing the source code files:

   ```
   $ cvs -z3 -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/bluefish \
                                   co bluefish-gtk2
   ```

   A lot of files will be downloaded, and listed one by one. If you're on dial-up, this might take a bit of time. When the downloads have completed, you will find the bluefish sources in the subdirectory `bluefish-gtk2`.
4. You can now enter that directory and install bluefish using the instructions in the section called “Installing from Development Source Tree”.

[1]Concurrent Versions System
[2]http://www.sourceforge.net

Chapter 2. Installing Bluefish

Requirements

Bluefish aims to be portable; that is, wherever GTK is ported. A comparatively small set of external libraries are necessary for it to work. Any recent GNU/Linux distribution or other *NIX with GTK2 installed should be sufficient. In addition to the list of requirements below, you may also want to look at the section called “System Specific Installation Issues”. Note that these requirements fit the GTK2 version. If you only have GTK1, you want the last GTK1-version, v0.7.

The main requirements:

- gtk v2.0
- libpcre

Optional requirements:

- gnome_vfs - *for remote file support*
- libaspell - *spell checker*
- grep & find - *used by the the section called “Open advanced”* dialog.

Compiling Bluefish requires a few additional packages. However, binary packages exist for many platforms, so it is likely you will not need to compile. Now, let us assume you want to compile, perhaps to get the latest and greatest from CVS. The requirements are as follows:

- Development files (header files, etc) for the packages above. *These are often distributed as separate packages. There is also a high probability you have these installed already.*
- gcc - *Bluefish has been tested to compile on the 2.95 and 3.x branches.*
- gmake or BSD make
- autoconf - *only if you are going to compile from CVS*

Quick Standard Installation

There are two main methods for installing Bluefish: Compile from source or install a binary package. Binary installation is easiest, so we will cover that first. There are a few different approaches, caused by the differences between systems. We will start off by summarizing a few really quick and simple approaches before dealing with this problem more extensively.

- Debian: **`su - && apt-get update && apt-get install bluefish`**
- Red Hat, Mandrake (and other Linux distributions that support rpm): Download the latest .rpm from the Bluefish website
- FreeBSD, NetBSD and OpenBSD distribute Bluefish through their packaging systems.
- To compile, or install on another platform, see the section called “Installing a Bluefish Source Distribution”.

System Specific Installation Issues

Different systems have different approaches to solutions and packaging. You might find the information below interesting.

Mandrake:

- libpcre: Breaks pcre into 3 different pieces, make sure pcre-devel is installed if compiling from source. Try this command:

  ```
  $ rpm -ql pcre-devel
  ```

- *... more nags with Mandrake?*

Debian:

- Debian Woody (the current Stable) has an old GTK-2 version, that contains several known bugs, but they are not serious.
- Debian Sarge (currently in Testing) has Bluefish 0.13, and a very recent GTK version.
- Debian Sid (Unstable) will always have the latest Bluefish version.

Installing a Bluefish Source Distribution

By installing Bluefish from source, you may be able to get a newer version (from CVS) than those distributed as binaries. You may also need to compile from source if no binary is available for your system.

Quick Installation Overview

This is the short installation description. Consult the other chapters if you are in doubt.

Bluefish is installed using the standard 'configure, make, make install' steps. Assuming you have downloaded a bluefish source package, for instance `bluefish-ver.tar.gz` (naturally, change the filename to what's appropriate), you complete the installation with the following steps:

1. **`tar -zxvf bluefish-ver.tar.gz`**
2. **`cd bluefish-ver`**
3. **`./configure`**
4. **`make`**
5. **`su -c 'make install'`**
6. Now, type **`bluefish`** to run. You may delete the `bluefish-ver` directory.

The **configure** script is used to automatically find the appropriate settings for your system. Because of differences between systems, this compile-time configuration is necessary, and configure solves this challenge easily -- with an added bonus of telling whether you have everything needed to compile.

The configure-script can be, um, configured. This is something you most likely will not need to do, but it is easy to do if necessary. For a complete list of configure options, see the section called “Configure Options”

Installing from Development Source Tree

You can get the latest Bluefish version via CVS using the instructions in the section called “Latest Developmental Version”. Next, install it with the following steps:

1. Enter the directory containing the bluefish source files: **`cd bluefish-gtk2`**
2. Next, generate the configure script by running **`autoconf`**
3. Then, you run **configure** with whatever options you might want.

   This example will cause **make install** to install Bluefish with the specified directory as prefix (i.e. the binary is installed in `/usr/local/bf-cvs/bin/bluefish`). This is most likely not what you want -- just run configure without parameters instead.

   ```
   $ ./configure --prefix=/usr/local/bf-cvs
   ```

   If configure fails, it will probably give a hint telling you what is missing or wrong.
4. Assuming it completed successfully, your next step is to compile Bluefish. To do this, run **`make`**.
5. When **make** has completed, you can install Bluefish: (**su** to root first, unless you specified a user writable prefix to configure), then issue: **`# make install`**.

To update the sources at a later time, you run the command **`cvs -z3 -q update`** from within the `bluefish-gtk2` directory.

Problems Compiling?

If compilation fails, first make sure you have the necessary utilities and libraries. See the section called “Requirements”.

Next, see if your system is mentioned in the section called “System Specific Installation Issues”.

Below is a list of well known problems that have been mentioned on the bluefish-dev list:

- make: *** No targets specified and no makefile found. Stop.

  This will happen if *configure* fails and you try to run **make**. It also happens if you're running **make** from the wrong directory.
- ... more trouble to come ;-)

If you're unable to find a solution (or if you think you have a solution others might want), feel free to contact us on the *bluefish-dev* list (See the section called “Contact Us”).

Configure Options

This section describes all the configure options available for bluefish.

Standard configuration flags

Configuration:

`-h, --help`
: display this help and exit

`--help=short`
: display options specific to this package

`--help=recursive`
: display the short help of all the included packages

`-V, --version`
: display version information and exit

`-q, --quiet, --silent`
: do not print "checking..." messages

`--cache-file=FILE`
: cache test results in FILE [disabled by default]

`-C, --config-cache`
: alias for `--cache-file=config.cache`

`-n, --no-create`
: do not create output files

`--srcdir=DIR`
: find the sources in DIR [configure dir or `..` by default]

Installation directories:

`--prefix=PREFIX`
: install architecture-independent files in PREFIX [`/usr/local` by default]

`--exec-prefix=EPREFIX`
: install architecture-dependent files in EPREFIX [`PREFIX` by default]

Tip
: By default, **make install** will install all the files in `/usr/local/bin`, `/usr/local/lib`, etc. You can specify an installation prefix other than `/usr/local` using `--prefix`, for instance `--prefix=$HOME`.

Fine tuning of the installation directories:

Tip
: For better control, use the options below. Defaults are shown within brackets.

`--bindir=DIR`
: user executables [`EPREFIX/bin`]

`--sbindir=DIR`
: system admin executables [`EPREFIX/sbin`]

`--libexecdir=DIR`
: program executables [`EPREFIX/libexec`]

`--datadir=DIR`
: read-only architecture-independent data [`PREFIX/share`]

`--sysconfdir=DIR`
: read-only single-machine data [`PREFIX/etc`]

`--sharedstatedir=DIR`
: modifiable architecture-independent data [`PREFIX/com`]

`--localstatedir=DIR`
: modifiable single-machine data [`PREFIX/var`]

`--libdir=DIR`
: object code libraries [`EPREFIX/lib`]

`--includedir=DIR`
: C header files [`PREFIX/include`]

`--oldincludedir=DIR`
: C header files for non-gcc [`/usr/include`]

`--infodir=DIR`
: info documentation [`PREFIX/info`]

`--mandir=DIR`
: man documentation [`PREFIX/man`]

Program names:

`--program-prefix=PREFIX`
: prepend PREFIX to installed program names

`--program-suffix=SUFFIX`
: append SUFFIX to installed program names

`--program-transform-name=PROGRAM`
: run sed PROGRAM on installed program names

System types:

`--build=BUILD`
: configure for building on BUILD [guessed]

`--host=HOST`
: cross-compile to build programs to run on HOST [BUILD]

Some influential environment variables:

Tip
: Use these variables to override the choices made by **configure** or to help it to find libraries and programs with nonstandard names/locations.

`CC`
: C compiler command

`CFLAGS`
: C compiler flags

`LDFLAGS`
: linker flags, e.g. -L<lib dir> if you have libraries in a nonstandard directory <lib dir>

`CPPFLAGS`
: C/C++ preprocessor flags, e.g. -I<include dir> if you have headers in a nonstandard directory <include dir>

`CPP`
: C preprocessor

Flags personal to bluefish

Optional Features:

Note
: It works as is: `--enable-feature` enables the feature, `--disable-feature` or `--enable-feature=no` disables the feature.

: By default, the `--enable-feature` option is not enabled, you should pass it if you want to get it, the `--disable-xxx` option is not disabled, you should pass it if you want to disable it.

`--enable-auto-optimization`
: Optimizes the build process for a given architecture if possible. It works only on a selected set of x86 platforms.

: How: rely on the result of:

1. **uname -p** or **grep "model name" /proc/cpuinfo | cut -d: -f2** to detect the architecture
2. the version of gcc to pass the arguments

: Tested gcc versions: 3.2.*, 3.0.*, 2.95.*

: Machines: Intel(R) Pentium(R) 4CPU, Pentium III, AMD-K6 (tm) 3D, Pentium 75 - 200, Pentium II, AMD Athlon(TM) XP

: Other machines are ignored

`--enable-gcc3-optimization`
: optimizes the build process for a given architecture if possible

: Machines: i386, i486, pentium, pentium-mmx, pentiumpro, pentium2, pentium3, pentium4, k6, k6-2, k6-3, athlon, athlon-tbird, athlon-4, athlon-xp, athlon-mp, winchip-c6, winchip2, c3

: Other machines are ignored

`--enable-gcc2-optimization`
: optimizes the build process for a given architecture if possible

  Machines: i386, i486, pentium, pentiumpro, k6

  Other machines are ignored

`--enable-debugging-output`
: turns debugging output on (this option impacts performance)

`--disable-splash-screen`
: suppresses the display of the splash screen at launch time (Bluefish launches faster)

`--enable-highlight-profiling`
: outputs statistics on where the program spends most of its time when highlighting patterns.

  Usage: for debugging highlight patterns or trying to optimize the program

`--enable-development`
: enables development checks (slows down the program)

`--enable-gprof-profiling`
: outputs statistics on where the program spends most of its time by generating extra code to write profile information suitable for the analysis. (slows down the program)

`--enable-gcoc-coverage`
: Purpose: to be able to collect statistics on how many times each branch is executed and how long it has lasted. Creates data files for the gcov code-coverage utility. (slows down the program)

`--disable-nls`
: disables the Native Language Support (might speed up the program)

Optional Packages:

Note
: This works as is: `--with-xxx=foo` enables the flag, `--without-xxx` disables it. When not enabled, the default is used.

`--with-gnome1-menu`
: customized path for the gnome1 menu.

  Usage: `--with-gnome1-menu=customizedpath` `--without-gnome1-menu`

  By default disabled.

`--with-freedesktop_org-menu`
: customized path for the freedesktop.org (gnome and kde) menu

  Usage: `--with-freedesktop_org-menu=customizedpath` `--without-freedesktop_org-menu`

  Defaults to autodetection. Autodetection will try:

  - `/usr/share/applications`
  - `PREFIX/share/applications`
  - `/usr/X11R6/share/gnome/applications`
  - `PREFIX/share/gnome/applications`

`--with-freedesktop_org-mime`
: customized path for the freedesktop.org (gnome and kde) mime

  Usage: `--with-freedesktop_org-mime=customizedpath` `--without-freedesktop_org-mime`

  Defaults to autodetection. Autodetection will try:

  - `/usr/share/mime`
  - `PREFIX/share/mime`
  - `/usr/X11R6/share/gnome/mime`
  - `PREFIX/share/gnome/mime`

`--with-gnome2_4-mime`
: customized path for the gnome 2.4 mime

  Usage: `--with-gnome2_4-mime=customizedpath` `--without-gnome2_4-mime`

  Defaults to autodetection. Autodetection will try:

  - `/usr/share/mime-info`
  - `PREFIX/share/mime-info`
  - `/usr/X11R6/share/gnome/mime-info`
  - `PREFIX/share/gnome/mime-info`

`--with-gnome2_4-appreg`
: customized path for the gnome 2.4 application registry

  Usage: `--with-gnome2_4-appreg=customizedpath` `--without-gnome2_4-appreg`

  Defaults to autodetection. Autodetection will try:

  - `/usr/share/application-registry`
  - `PREFIX/share/application-registry`
  - `/usr/X11R6/share/gnome/application-registry`
  - `PREFIX/share/gnome/application-registry`

`--with-icon-path`
: customized path for the icon.

  Usage: `--with-icon-path=customizedpath` `--without-icon-path`

  Defaults to auto detection. Autodetection will try:

  - `/usr/share/pixmaps`
  - `PREFIX/share/pixmaps`
  - `/usr/X11R6/share/gnome/pixmaps`
  - `PREFIX/share/gnome/pixmaps`

`--with-libiconv-prefix`
: customized path for libiconv top level installation.

  Usage: `--with-libiconv-prefix=customizeddir`

  Effect: searches for libiconv in `customizeddir/include` and `customizeddir/lib`

`--with-included-gettext`
: use the GNU gettext library included in the package

Installing a Binary Distribution

Different packages -- different installation. We will cover only a few approaches here[1], since the installation is very system-specific ;-). Let us have a look at some different systems:

For Debian users this is very simple. To download, install, and configure bluefish in “One Swift Move”, run:

```
$ su - && apt-get update && apt-get install bluefish
```

You can check if the version available through apt is the latest -- see the Bluefish homepage, and compare the version there with what **`apt-cache show bluefish`** tells you. If there is a newer version on the Bluefish site, download it and install the package like this: **`dpkg -i bluefish-ver.deb`**

For rpm based distributions, first check if your distribution has a recent Bluefish version. If it does not, download the rpm for your distribution from any of the Bluefish mirrors. Installing a downloaded rpm is as simple as pointing and clicking in your favorite GUI package manager, or issuing the following command from the command line (as root):

```
# rpm -Uvh bluefish-ver.rpm
```

If you're using FreeBSD, NetBSD or OpenBSD, we probably do not need to tell *you* how to use your favorite package system ;-)

For Mac users, just install it via Fink.

Post-installation Setup

The first time you run Bluefish it will create a directory `~/.bluefish` where all Bluefish's configuration options are stored. This includes all preferences, customized menus, highlighting-patterns, file history, etc.

Bluefish will work right out of the box, but you can and should take advantage of the many customizations available. Change the font in the main text view if you do not like it, remove unused tool bars, add shortcuts to the customizable menu, and edit the list of browsers and external programs.

If you are upgrading from a previous version, perhaps CVS, you should note that the syntax highlighting may have changed.

[1]If you want to contribute a description on how to install Bluefish on your system, just drop us a note. :-)

To make sure you have the latest highlighting patterns, follow the following procedure:

Procedure 2.1. Getting the new defaults after upgrading - First method

1. Exit Bluefish
2. Delete the `highlighting` file in your `~/.bluefish` directory.

   Next time Bluefish is started, the new defaults will be loaded.

Note that this will also annihilate all your changes to the highlighting. Here's a more gentle approach:

Procedure 2.2. Getting the new defaults after upgrading - Second method

1. Exit Bluefish
2. Move your current highlighting file to highlighting.old
3. Start Bluefish to get the new patterns
4. Exit Bluefish
5. Run **`diff -c highlighting.old highlighting`** to find the differences.

If your settings become corrupted, unusable, or you simply want to revert to the defaults, you may safely delete the `~/.bluefish` directory.

Chapter 3. Using Bluefish

In this chapter, most of the functionalities of Bluefish are described. What you can do, how you do it, and how you can customize the default behavior.

Starting Bluefish

In GNOME, Bluefish can be started from the Applications/Programming menu. From a terminal, simply launch bluefish using the command bluefish.

Command line options

There are several useful command line options:

-s
: skip root check

-v
: display the current version

-n
: open a new window

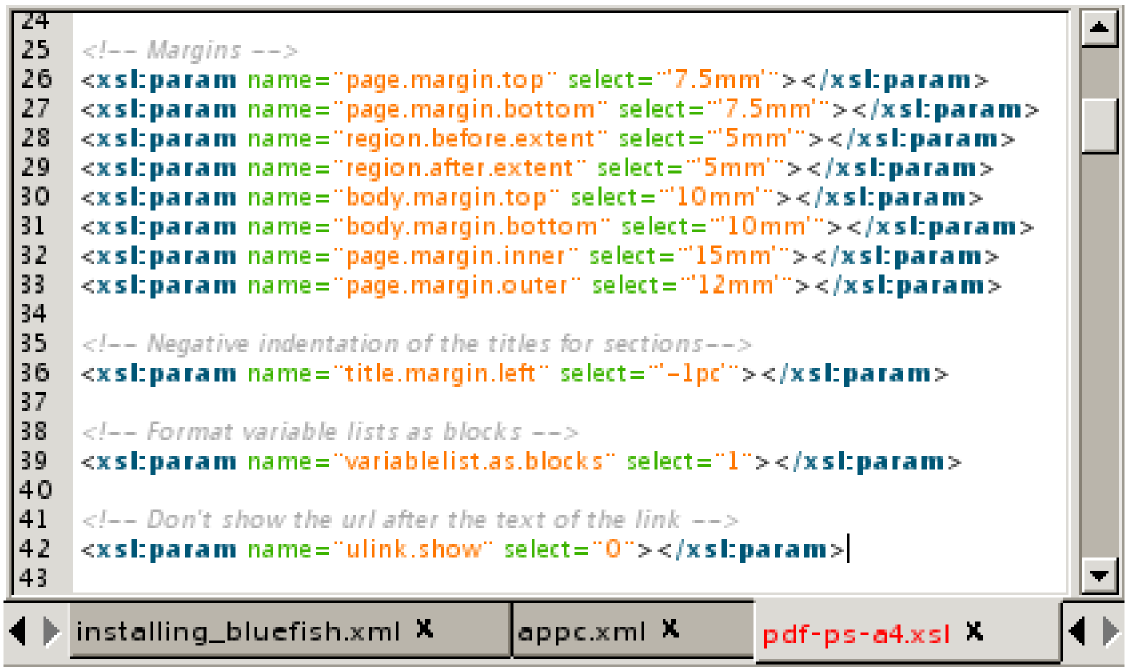
-p filename
: open a project

-h
: display this help screen

Many programs like browsers, email clients and file managers can be configured to open files in Bluefish. For example, **bluefish '%s'** will open a file in the current window, **bluefish -n '%s'** will open a file in a new window, and **bluefish -p '%s'** will open a project file.

The user interface

The biggest part of the user interface is the editor area. Because Bluefish has a so-called "Multiple Document Interface", there are actually many editor areas in Bluefish, accessible via the tabs. By default the tabs are on the bottom.



```
<!-- Margins -->
<xsl:param name="page.margin.top" select="'7.5mm'"></xsl:param>
<xsl:param name="page.margin.bottom" select="'7.5mm'"></xsl:param>
<xsl:param name="region.before.extent" select="'5mm'"></xsl:param>
<xsl:param name="region.after.extent" select="'5mm'"></xsl:param>
<xsl:param name="body.margin.top" select="'10mm'"></xsl:param>
<xsl:param name="body.margin.bottom" select="'10mm'"></xsl:param>
<xsl:param name="page.margin.inner" select="'15mm'"></xsl:param>
<xsl:param name="page.margin.outer" select="'12mm'"></xsl:param>

<!-- Negative indentation of the titles for sections-->
<xsl:param name="title.margin.left" select="'-1pc'"></xsl:param>

<!-- Format variable lists as blocks -->
<xsl:param name="variablelist.as.blocks" select="1"></xsl:param>

<!-- Don't show the url after the text of the link -->
<xsl:param name="ulink.show" select="0"></xsl:param>
```

Figure 3.1. Bluefish Editor Array

The top of the Bluefish interface consists of a menu, a main tool bar, an HTML tool bar, and a Custom menu.

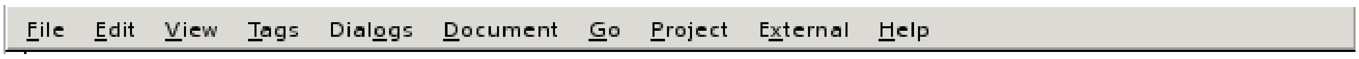


Figure 3.2. Bluefish Main Menu

The main tool bar gives you quick access to the basic functionalities of a text editor.

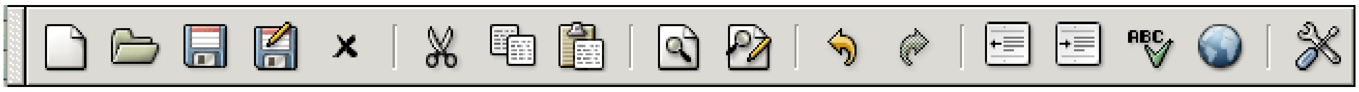


Figure 3.3. Bluefish Main Tool Bar

The HTML tool bar provides access to the most commonly used HTML functionalities.

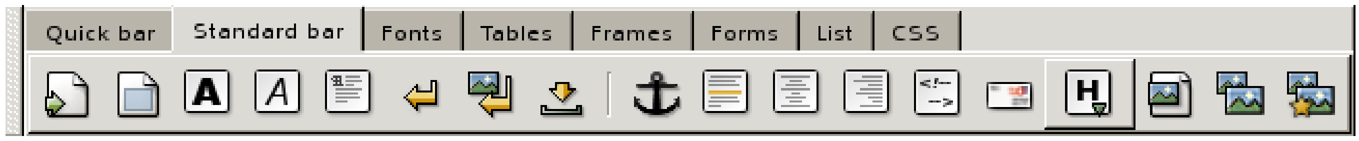


Figure 3.4. Bluefish HTML Tool Bar

The custom tool bar provides access to languages and replacement functions. It is fully customizable through the preferences panel.

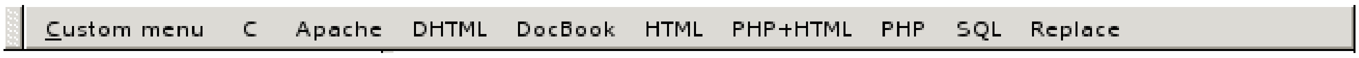


Figure 3.5. Bluefish Custom Tool Bar

To the left of the editor area is the side panel. If you would prefer that the side bar be on the right side, simply change the setting in the User Interface tab found in the Edit->Preferences menu option. The side panel consists of a file browser, a function reference browser, and a bookmark browser.

The file browser provides quick access to files and directories.

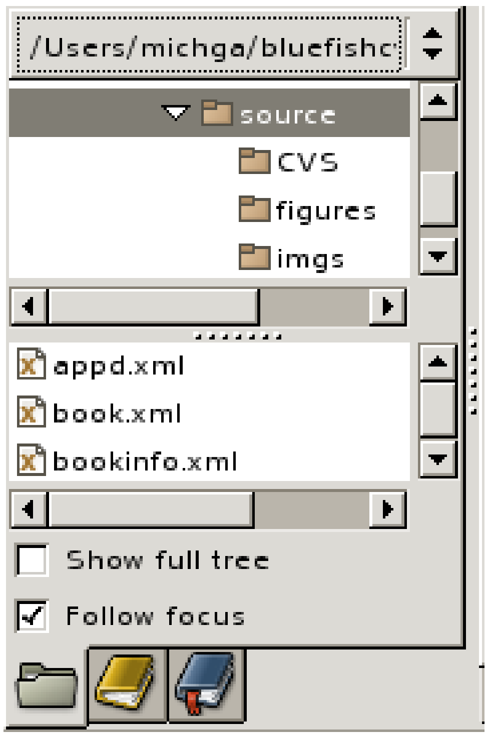
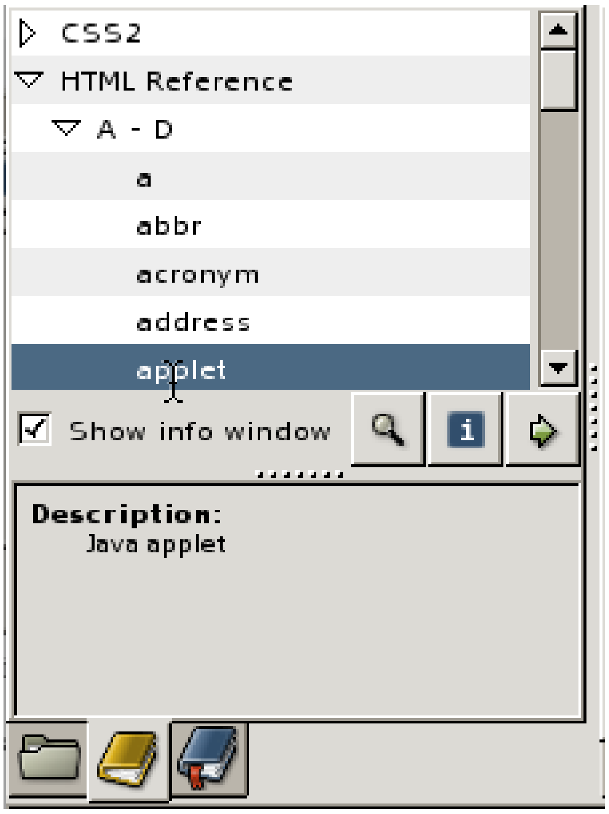


Figure 3.6. Bluefish File Browser

The function reference browser references CSS2, HTML, PHP, and Python functions with their syntax. Some of them provide dialogs to help you inserting them ,



Description:
Java applet

Figure 3.7. Bluefish Function Reference Browser

The bookmark browser provides access to previously marked positions in a file.

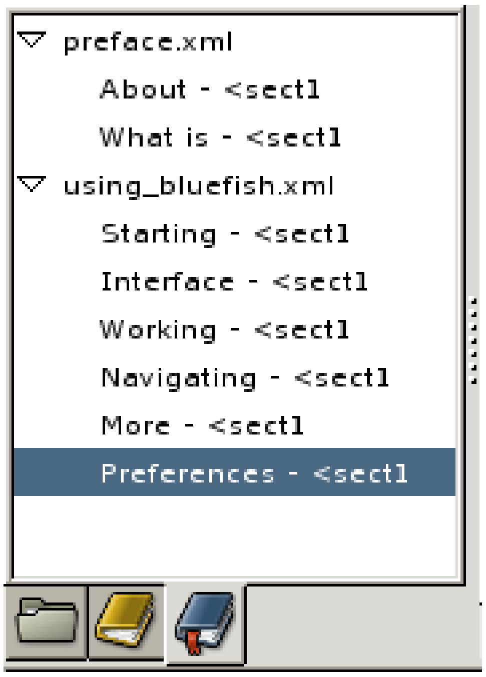


Figure 3.8. Bluefish Bookmark Browser

On the bottom of the Bluefish window is the status bar. Shown here are messages, the current line & column number, the insert (INS) or overwrite (OVR) mode for the cursor, and the file type & character encoding.

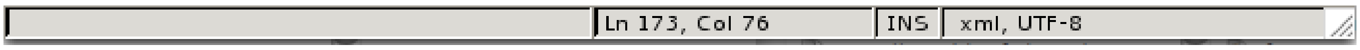


Figure 3.9. Bluefish Status Bar

The visibility of these items can be toggled via the View menu.

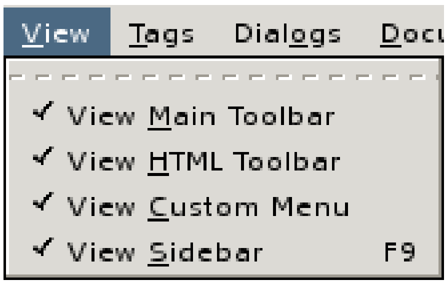
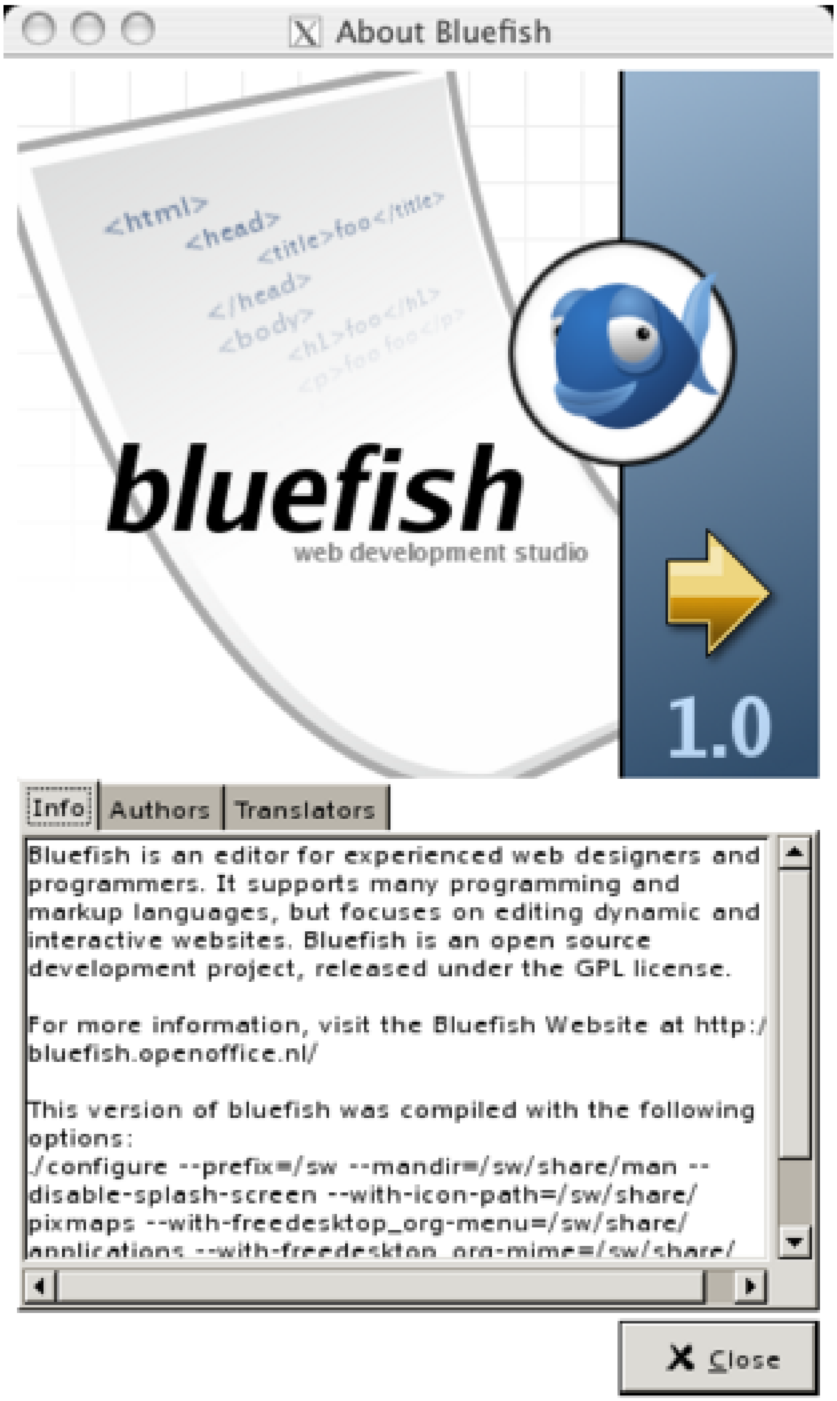


Figure 3.10. Bluefish View Menu

If you want to disable any of these items by default, you can set these options in the preferences under User interface.

The Help menu contains the typical About box. As usual, you will find in it developers', maintainers', and translators' details. Plus the configure flags used to compile Bluefish on your system.



Bluefish is an editor for experienced web designers and programmers. It supports many programming and markup languages, but focuses on editing dynamic and interactive websites. Bluefish is an open source development project, released under the GPL license.

For more information, visit the Bluefish Website at http:/ bluefish.openoffice.nl/

This version of bluefish was compiled with the following options:
./configure --prefix=/sw --mandir=/sw/share/man --disable-splash-screen --with-icon-path=/sw/share/pixmaps --with-freedesktop_org-menu=/sw/share/applications --with-freedesktop_org-mime=/sw/share/

Figure 3.11. Bluefish About Window

The other menus are described in the following sections:

- the Tags and Dialogs menu in the section called “HTML”
- the Go menu in the section called “Navigating through many documents”
- the Project menu in the section called “Projects”
- the External menu in the section called “External programs, filters”

Working with files

Most of the file operations are accessible from the File menu. Using this menu, a new file can be created, existing files opened, and opened files saved or renamed.

It is also possible to insert a file into another one, and to revert a modified file to its previously saved state.

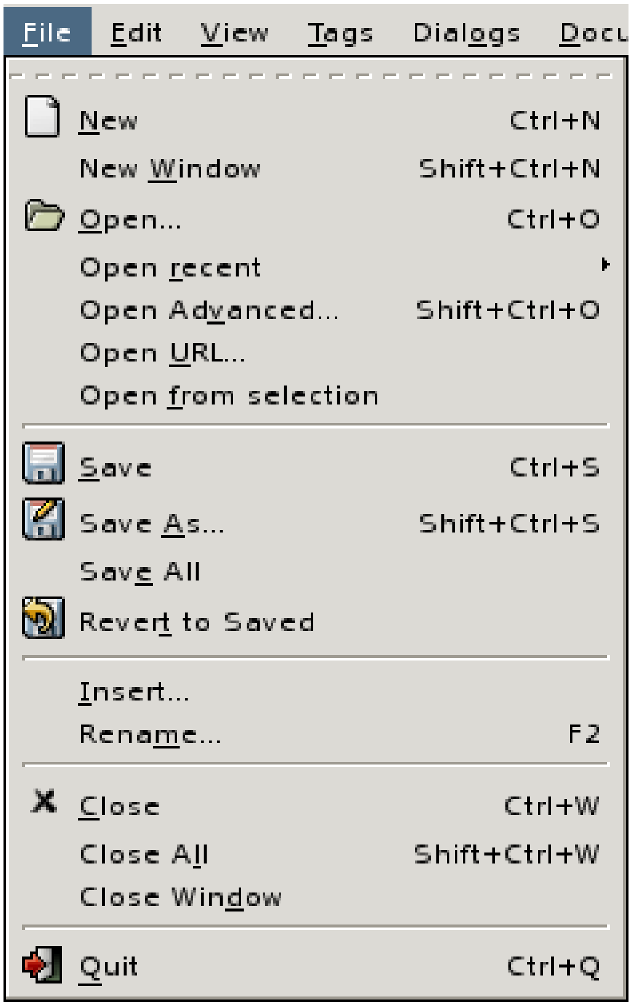


Figure 3.12. Bluefish File Menu

Creating files

Apart from using File->New (**Ctrl-N**) or the New icon to create a new file, you may also use File->New Window (**Shift-Ctrl-N**).

Both methods create an untitled file of type `text` with the default character encoding defined in the Files tab in the Edit->Preferences menu option.

Opening files

Through File->Open... (**Ctrl-O**), one or more files can be opened. When creating new files, you may want to open the files in a new window. In this case, use File->New Window to first open the new window and then open the desired files as usual.

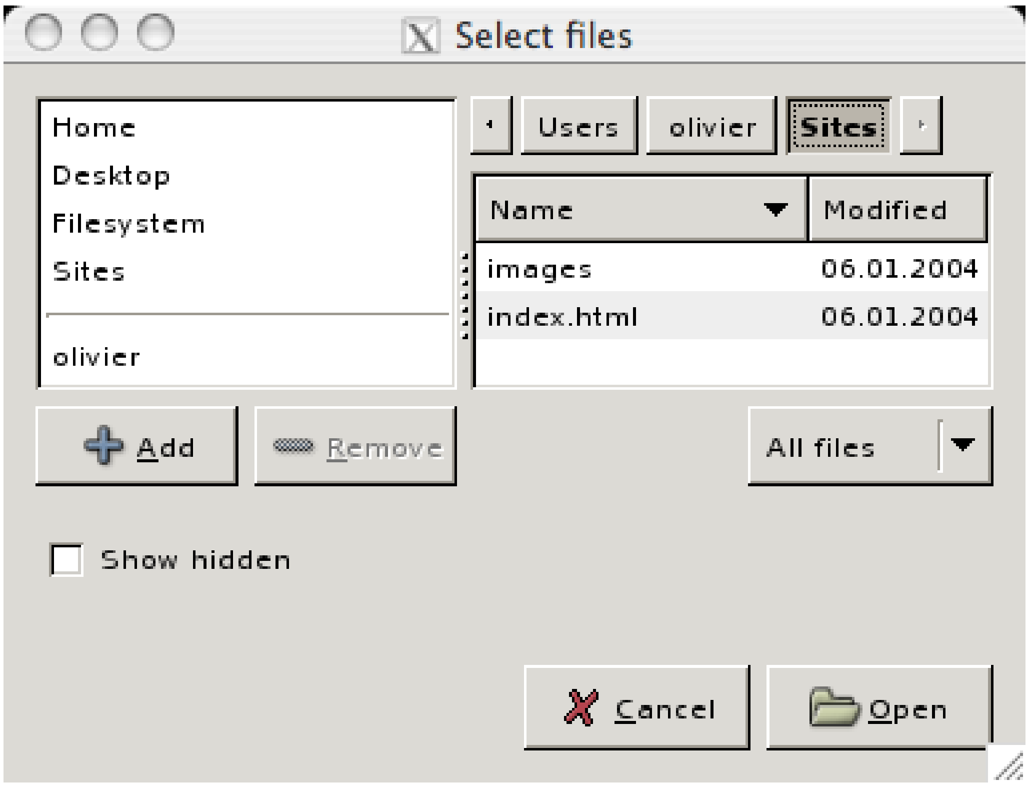


Figure 3.13. Bluefish Open File Dialog

Note

The most recently opened directories appear in the upper part of the left side panel, while the lower part contains user-defined locations. To add a new directory to the list, click on Add. You can also filter the file list by file type using the pop menu located on the right side. The list of file types in the filter menu is provided through the Filetypes tab found in Bluefish's Edit->Preferences menu option.

Recently opened files can be opened by selecting them from the list within File->Open recent. The number of files in this menu can be set in the preferences under Files.

The file browser in the side panel can also be used to open files. It supports filtering files, by right clicking the contextual menu in the file browser.

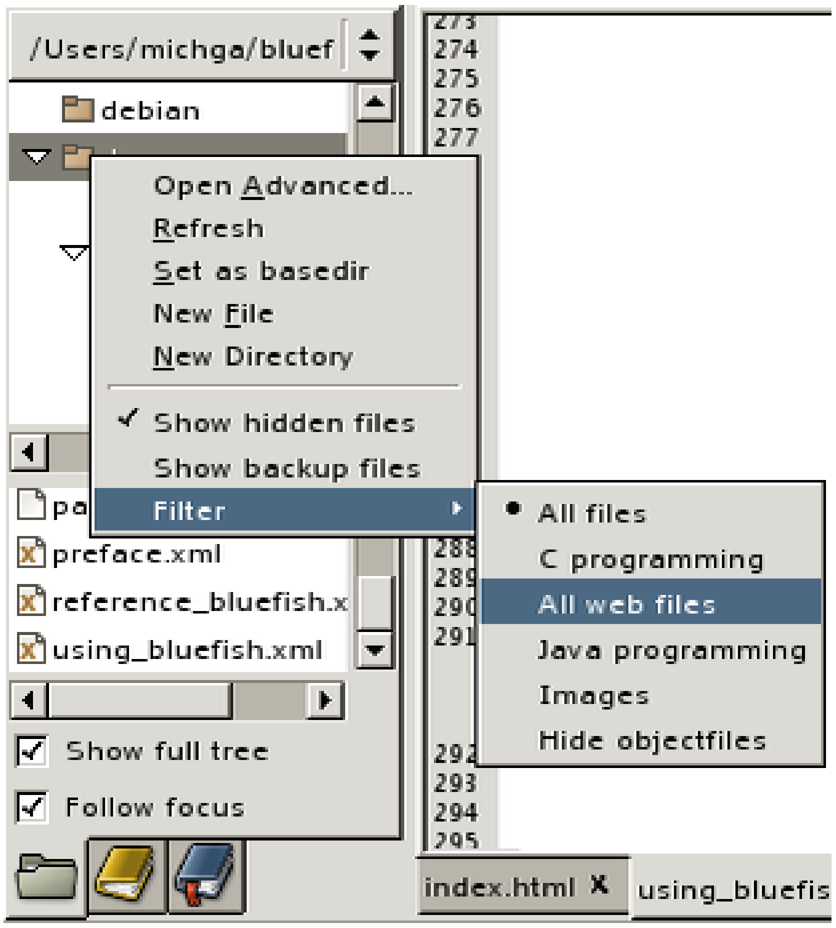


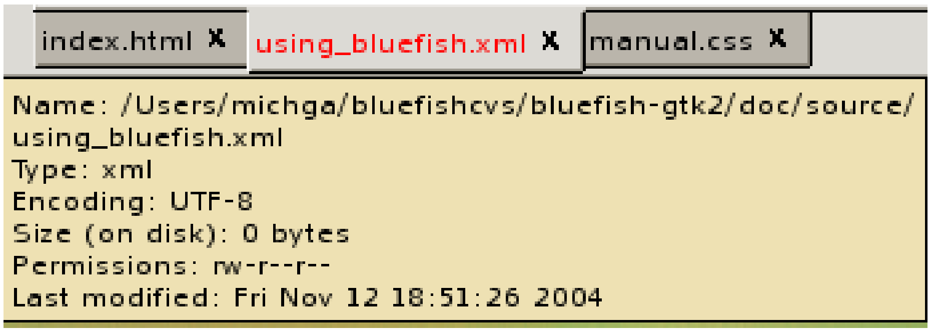
Figure 3.14. Filtering Files with the Bluefish File Browser

The available filters may be modified in Preferences. For more information, see the section called "Modifying the files filters".

If you right click a directory, you can make this directory the base directory for the file browser using the Set as basedir option. Then you can access it directly from the pop up menu in the upper part of the file browser.

By default the file browser follows the document focus. If you change to a different document, the file browser will show the contents of the directory where this document is located. This behaviour can be changed on the bottom of the file browser.

Information about currently opened files can be seen if you move the mouse over the document tab (by default on the bottom of the screen). A so called tool tip will be shown with information about the full path, size, permissions, file type and encoding of the file.



```
Name: /Users/michga/bluefishcvs/bluefish-gtk2/doc/source/
using_bluefish.xml
Type: xml
Encoding: UTF-8
Size (on disk): 0 bytes
Permissions: rw-r--r--
Last modified: Fri Nov 12 18:51:26 2004
```

Figure 3.15. Info on open file with the Bluefish File Browser

An interesting feature of Bluefish is the ability to open files by selecting the text of a currently opened file. For example, if a filename is shown in say a terminal application, you can select the filename, and use File->Open from Selection to open that file. The file, if it exists, will be opened in another tab within Bluefish.

Finally, files can be opened via the command line by feeding filenames to Bluefish as arguments. This can even be done while Bluefish is running and the resulting file will then show up in its own tab.

Files can also be opened by clicking on the Open... icon in the main tool bar.

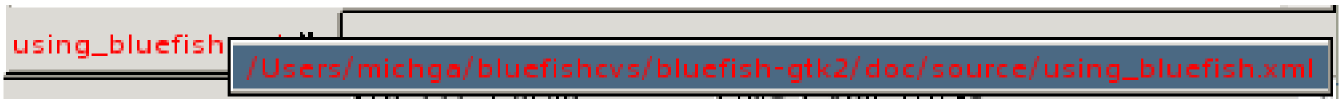
Note

> If you have installed gnome-vfs or gnome-vfs2 before installing Bluefish, you will be able to open files on remote desktop.

Be aware that if the file is huge it may take a very long time to get the rendering if syntax highlighting is enabled. The GTK editing widget used in Bluefish, furthermore, is not very good at handling files with very long lines, and that could also slow down Bluefish considerably.

Saving files

If a document is modified, the filename is shown in red in the document tabs, and also if you right click on the tabs, the full path is shown in red in the list that will pop up.



using_bluefish

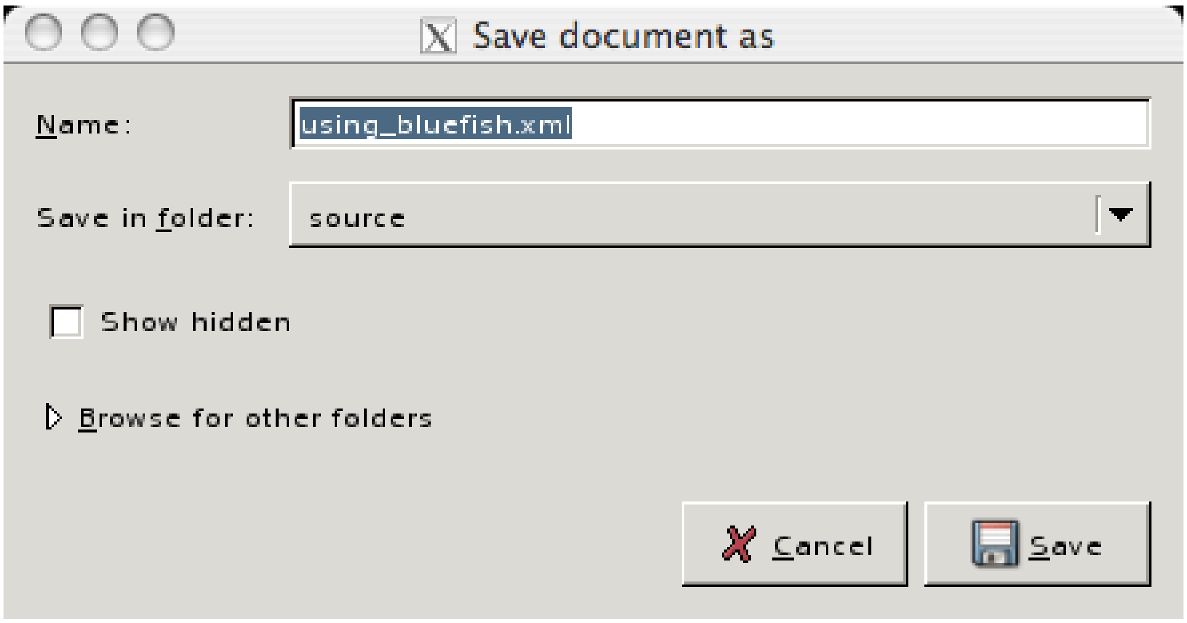
/Users/michga/bluefishcvs/bluefish-gtk2/doc/source/using_bluefish.xml

Figure 3.16. Tool Tip for Modified File

To save a document, you can use the File menu, the Save icon in the tool bar, or press the shortcut key combination **Ctrl-S**. By default a backup is made during save. The original file is copied to the same filename with a tilde ~ appended. This suffix and the backup behaviour can be changed in the preferences under Files.

Before saving the file, Bluefish will check if the original file was changed on disk, using the last modified time and the file size. On some filesystems the last modified time is sometimes not very precise (most notably on samba mounts). This makes Bluefish think the file is modified when it is not. This check can be changed in the preferences under Files.

You can also save a document under a different name, using the Save As... (**Shift-Ctrl-S**) menu entry, or the Save As... icon in the main tool bar. The original file will still exist.



Save document as

Name: using_bluefish.xml

Save in folder: source

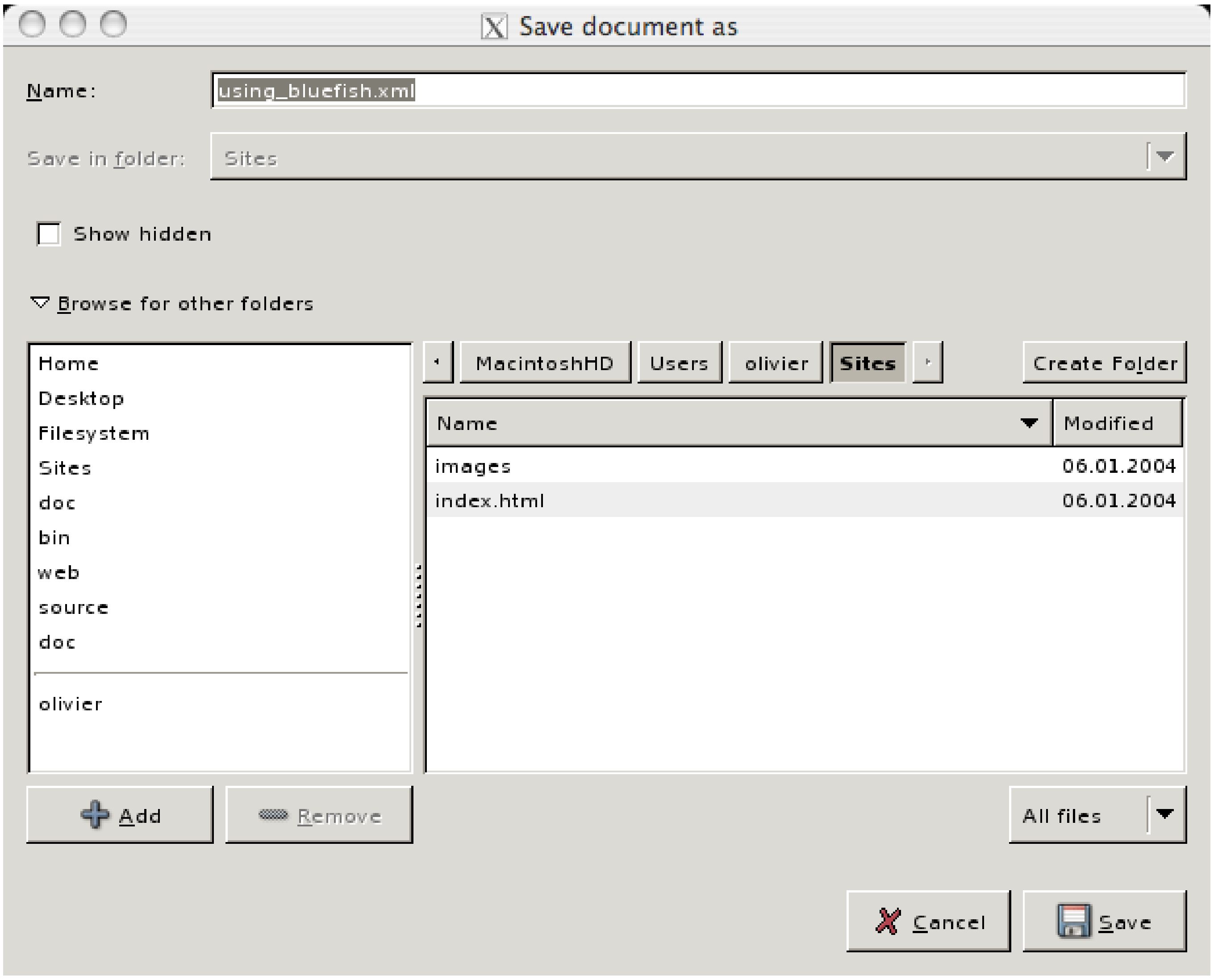
Show hidden

Figure 3.17. Saving a File under a new Name

To save all modified files, you can use the File->Save All menu entry. This will save all documents that have been modified and present you with a save dialog if some files are new files.

Renaming files

It is also possible to move or rename a document, using the File->Rename... (**F2**) menu item, or right-clicking the file name in the side panel and choosing the Rename item.



Name: using_bluefish.xml

Save in folder: Sites

Home
Desktop
Filesystem
Sites
doc
bin
web
source
doc
olivier

MacintoshHD | Users | olivier | Sites

Name	Modified
images	06.01.2004
index.html	06.01.2004

Figure 3.18. Moving a file to another location

Closing files

When you want to close a file quickly, click on the close icon in the document tab. You may also use the Close icon in the main tool bar, or the File->Close (**Ctrl-W**) menu item.

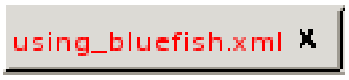
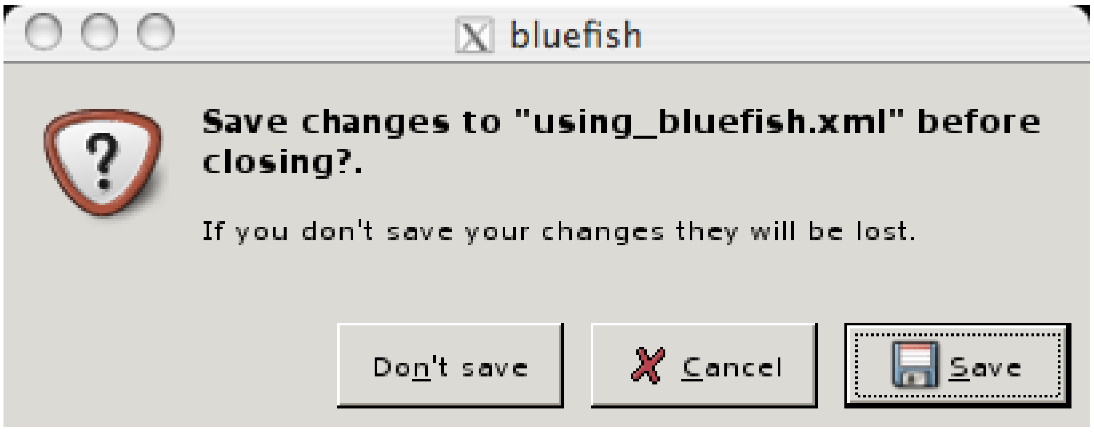


Figure 3.19. Closing a file with the document tab icon

If the file is unchanged, it is merely closed. If the file has been modified, you will be presented with a save dialog.



bluefish

Save changes to "using_bluefish.xml" before closing?.

If you don't save your changes they will be lost.

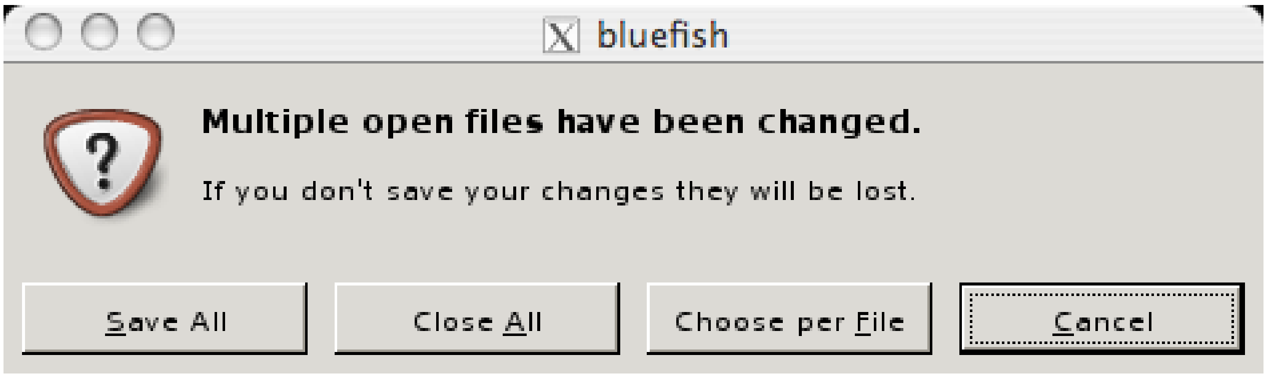
Figure 3.20. Closing a modified file

Tip

Use it to save and close a file in one step.

When dealing with multiple files, you may want to use the File->Close All (**Shift-Ctrl-W**) menu item.

For each modified file, you will be presented with a save dialog, where you can choose to save the changes, close the file (i.e. discarding any change), or cancel the operation.



bluefish

Multiple open files have been changed.

If you don't save your changes they will be lost.

Figure 3.21. Closing all files

Tip

Say you have a number of open files, and only a few of them have been changed. To quickly close the unchanged files, and remain with the modified ones, use it answering cancel for the latter ones.

Note that the File->Close Window menu item offers the same behaviour.

Inserting files

You can insert any file into the current document with the File->Insert... menu item. The file will be inserted at the cursor location.

For more in-depth information about dealing with files, see the section called “More on files”.

Editing

Undo and Redo

The undo and redo functionalities are available from the Edit menu, the main tool bar, and the keyboard shortcuts.

- Undo (**Ctrl-Z**)
- Redo (**Shift-Ctrl-Z**)

The functions Undo All and Redo All in the Edit menu will undo or redo all of the stored changes. The maximum number of changes can be configured in the preferences, by default Bluefish will remember the last 100 changes per document. It is possible to clear the changes after the document is saved, an option in the preferences which is disabled by default.

Cut, Copy, and Paste

The functions Cut, Copy, and Paste are available from the Edit menu, the main tool bar, and the keyboard shortcuts.

- Cut (**Ctrl-X**)
- Copy (**Ctrl-C**)
- Paste (**Ctrl-V**)

On X Windows Systems, you can also paste the current selected text using the middle mouse button. First select some text (in Bluefish or in any other X application), then press the middle mouse button where you want to paste the selected text.

Cut or copy and then paste can also be done by selecting some text and dragging it to the destination. If the text is dragged to another document (or another application), it is copied. If the text is dragged within one document it is moved. Dragging highlighted text from one application to another may or may not work. However, most GNOME and GTK programs support this feature.

Input methods

Bluefish handles a number of input methods, available from the contextual menu within a given document.

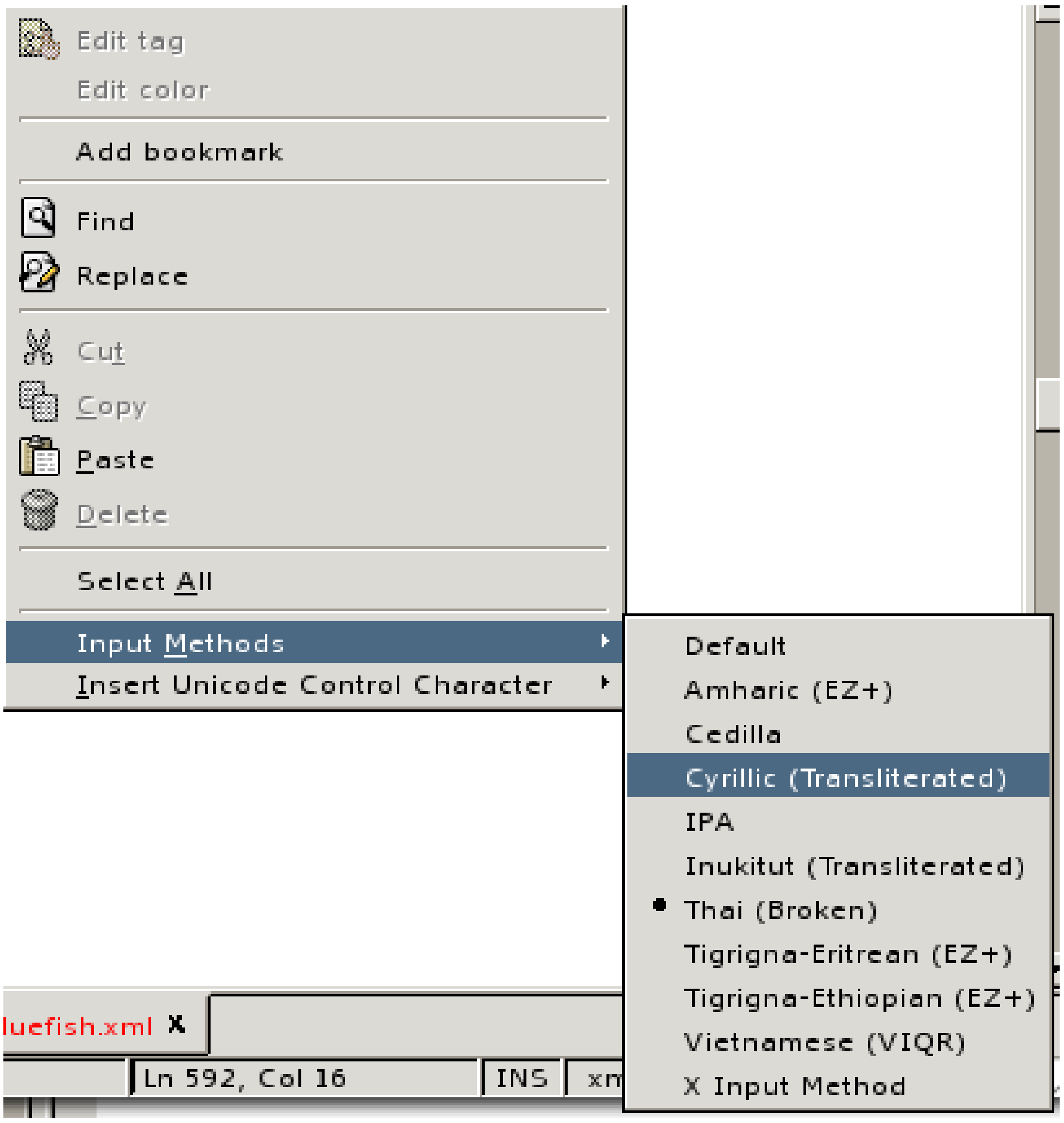


Figure 3.22. The Input Methods Contextual menu

- The default mode switches all input methods off.
- The Amharic mode is used for the most popular Ethiopian language.
- The Cedilla mode is used for languages such as French, which uses the cedilla.
- The Cyrillic mode is used to enter Russian with Roman letters. The transliteration occurs immediately.
- The Inukitut mode works the same as Cyrillic mode.

- The IPA mode is used for International phonetic alphabet.
- Other modes are used for Erythrean, Ethiopian, Thai and Vietnamese languages.

The X Input method relies on a client-server input system, and an input server.

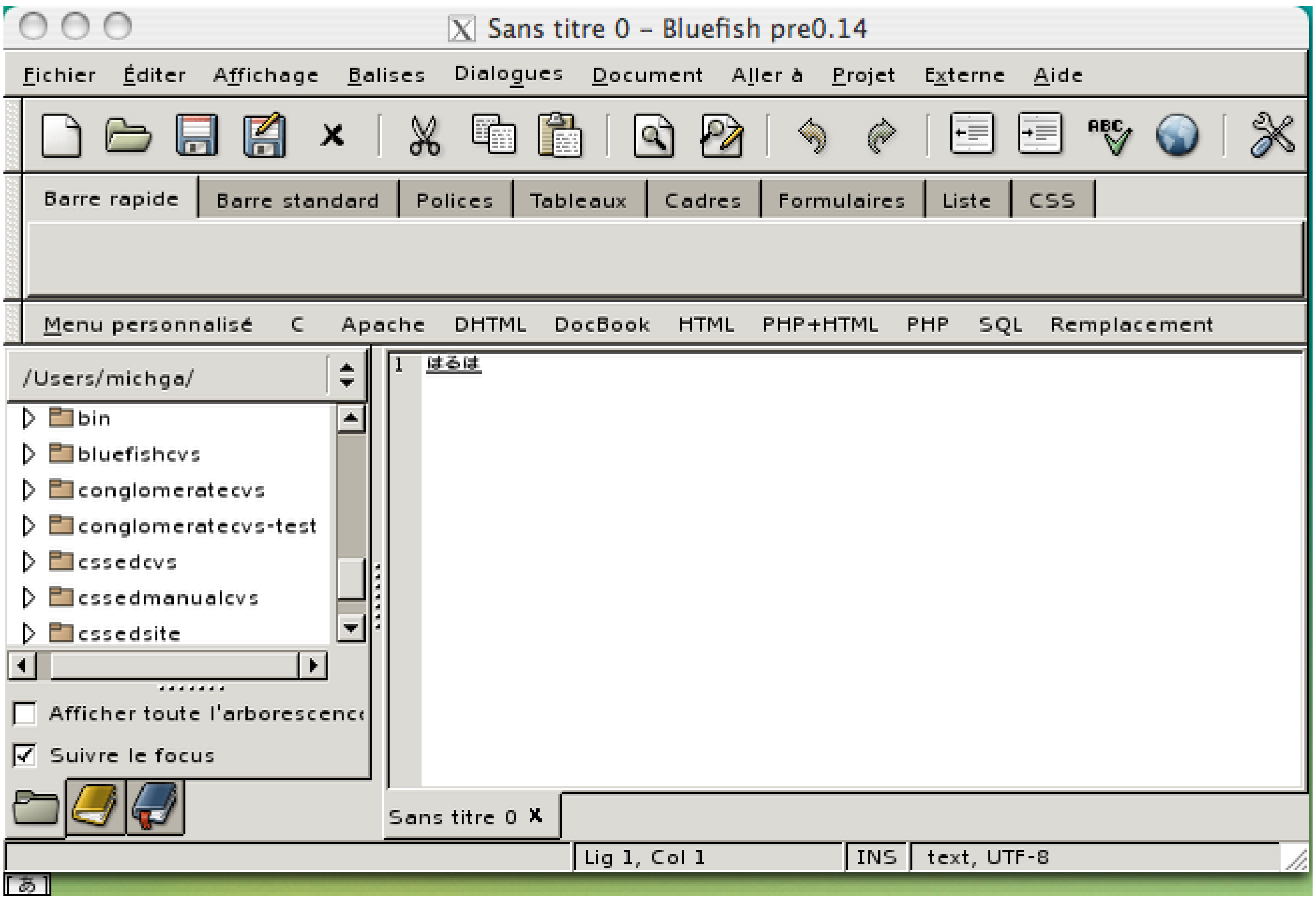
For Japanese, Chinese, and Korean documents, you may have to install and launch the correct input system, such as canna, and the appropriate input server, such as kinput2.

Here is how to write a Japanese document on a non-Japanese system.

Procedure 3.1. Writing in Japanese with Bluefish on a non-Japanese system

1. Launch the canna server if it is not running already
2. Set the encoding to Japanese, for example: **`export LANG=ja_JP.UTF-8`**
3. Set the Xinput method with **`export XMODIFIERS="@im=kinput2"`**
4. Launch kinput2 as a background process with **`kinput2 &`**
5. Launch bluefish as a background process with **`bluefish &`**
6. To activate the Xinput method within bluefish, use **Shift-Space**. A small window with a Japanese glyph will appear at one of the corner of the Bluefish window. Once the desired glyph has been composed, press **Space**, and hit **enter** to validate it.

Here, you can see the small Xinput method window, at the bottom left corner of the window and the first Japanese word not already validated in the Bluefish window launched on a French system.


Sans titre 0 - Bluefish pre0.14
Fichier Éditer Affichage Balises Dialogues Document Aller à Projet Externe Aide
Barre rapide Barre standard Polices Tableaux Cadres Formulaires Liste CSS
Menu personnalisé C Apache DHTML DocBook HTML PHP+HTML PHP SQL Remplacement
/Users/michga/
bin
bluefishcvs
conglomeratecvs
conglomeratecvs-test
cssedcvs
cssedmanualcvs
cssedsite
Afficher toute l'arborescence
Suivre le focus
Sans titre 0
Lig 1, Col 1
INS
text, UTF-8
あ

1 はるは

Figure 3.23. Writing in Japanese with Bluefish

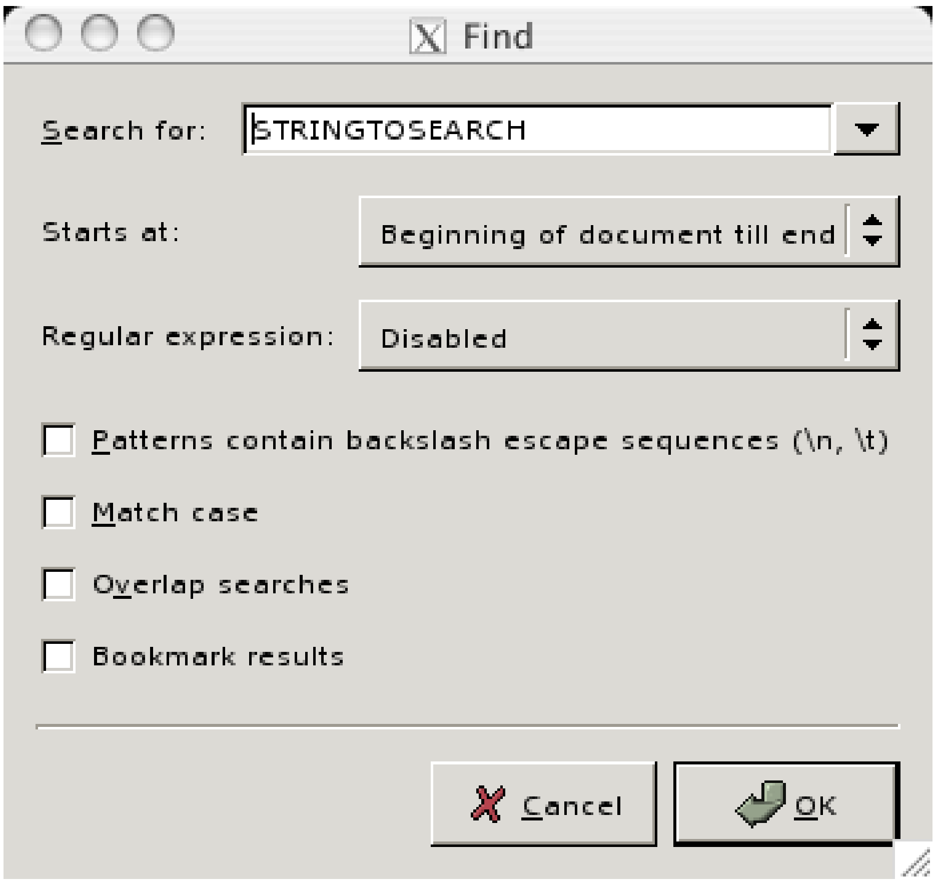
For an in-depth discussion on that subject, see Inputting from the keyboard.

Basic Find and Replace

Bluefish offers a wide range of find and replace methods in the Edit menu, also available through the contextual menu within a document. Here we will explore the most basic ones. For advanced find and replace methods, see the section called “Find and Replace”.

Searching for a word within a whole document

Choose the Edit->Find... (**Ctrl-F**) menu item. A Find dialog will be displayed. Enter the word to search for in the Search for: field. Then click OK.


Find

Search for: STRINGTOSEARCH

Starts at: Beginning of document till end

Regular expression: Disabled

☐ Patterns contain backslash escape sequences (\n, \t)

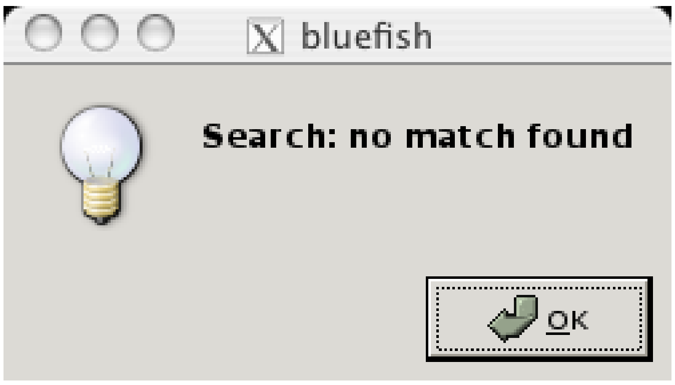
☐ Match case

☐ Overlap searches

☐ Bookmark results

Figure 3.24. Finding a word in a document, from start to end

If the word does not exist in the document, a small window pops up.

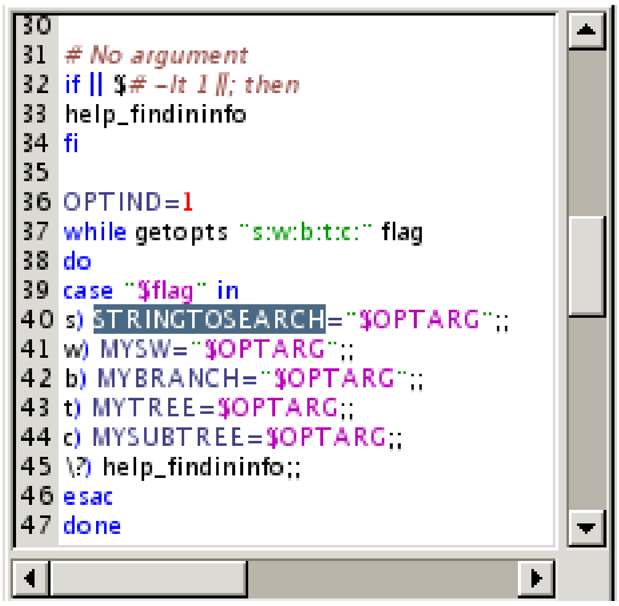

bluefish

Search: no match found

Figure 3.25. Unsuccessful search window

If the search is successful, the document window scrolls up to the first occurrence of the string in the document and highlights it.

Below is an example of a search applied to a shell script.



```
# No argument
if [[ $# -lt 1 ]]; then
help_findininfo
fi

OPTIND=1
while getopts "s:w:b:t:c:" flag
do
case "$flag" in
s) STRINGTOSEARCH="$OPTARG";;
w) MYSW="$OPTARG";;
b) MYBRANCH="$OPTARG";;
t) MYTREE=$OPTARG;;
c) MYSUBTREE=$OPTARG;;
\?) help_findininfo;;
esac
done
```

Figure 3.26. Highlighted search result in the document window

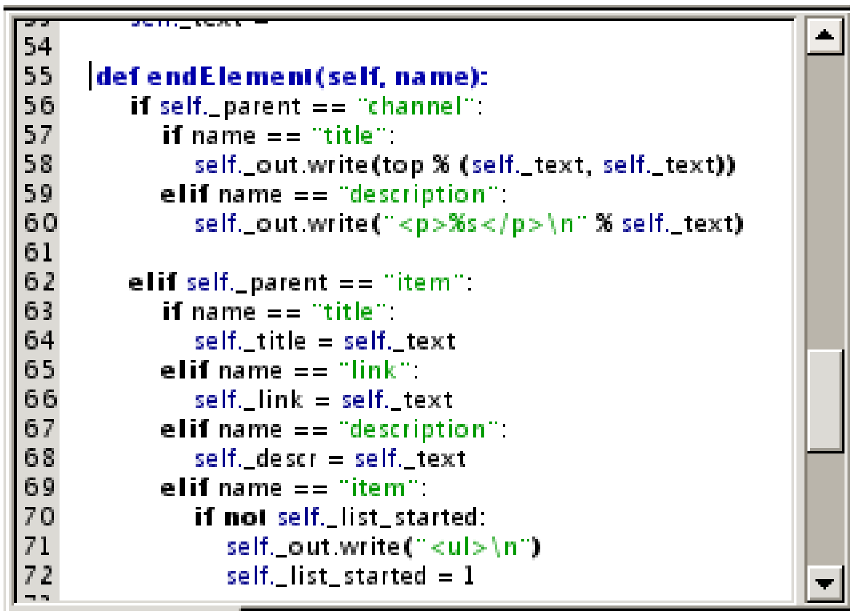
To find a subsequent occurrence of the string, use the Edit->Find again (**Ctrl-G**) menu item. If no further occurrence is found, a dialog will be displayed notifying you that no match was found.

Setting limits to the search scope

You may want to search for a string from the cursor location till the end of the document. Here is an example to search all `name ==` occurrences within a python script from a given location.

Procedure 3.2. Searching from selection

1. Put the cursor where you want to start the search from in the document window

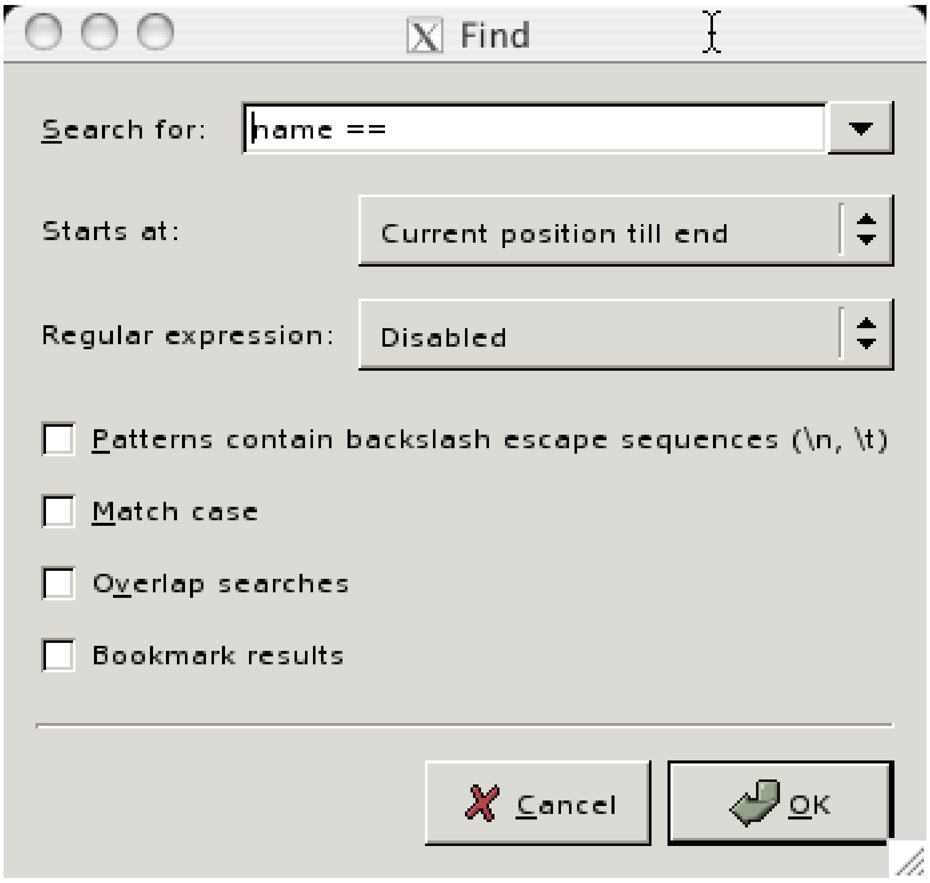


```
def endElement(self, name):
    if self._parent == "channel":
        if name == "title":
            self._out.write(top % (self._text, self._text))
        elif name == "description":
            self._out.write("<p>%s</p>\n" % self._text)

    elif self._parent == "item":
        if name == "title":
            self._title = self._text
        elif name == "link":
            self._link = self._text
        elif name == "description":
            self._descr = self._text
        elif name == "item":
            if not self._list_started:
                self._out.write("<ul>\n")
                self._list_started = 1
```

Figure 3.27. Setting the cursor location

2. Open the Find... dialog
3. Enter your search string in the Search for: field

4. Choose Current position till end from the Starts at: pop up menu

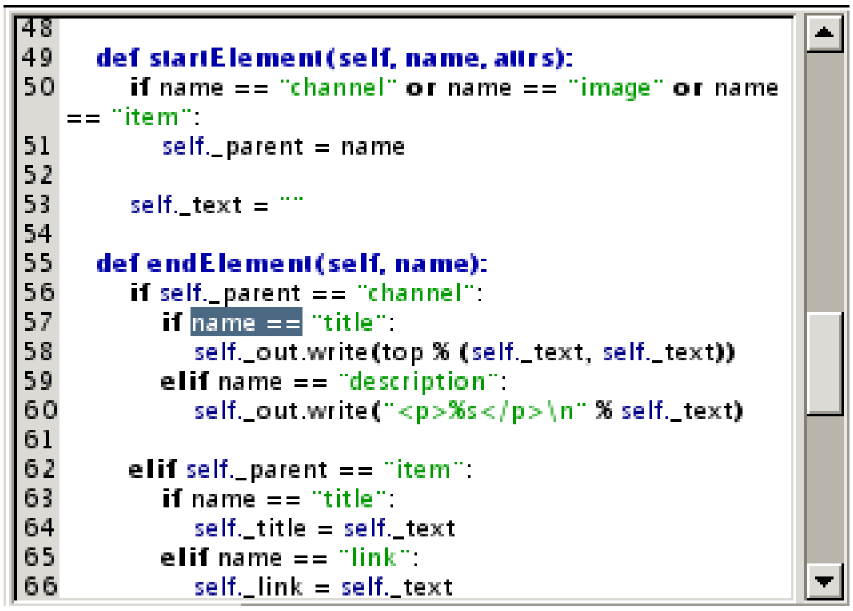


```
Find

Search for: name ==
Starts at: Current position till end
Regular expression: Disabled
[ ] Patterns contain backslash escape sequences (\n, \t)
[ ] Match case
[ ] Overlap searches
[ ] Bookmark results

Cancel    OK
```

Figure 3.28. Choosing a limited search method

5. Click OK.

Here is the result:



```
48
49      def startElement(self, name, attrs):
50          if name == "channel" or name == "image" or name
     == "item":
51              self._parent = name
52
53          self._text = ""
54
55      def endElement(self, name):
56          if self._parent == "channel":
57              if name == "title":
58                  self._out.write(top % (self._text, self._text))
59              elif name == "description":
60                  self._out.write("<p>%s</p>\n" % self._text)
61
62          elif self._parent == "item":
63              if name == "title":
64                  self._title = self._text
65              elif name == "link":
66                  self._link = self._text
```

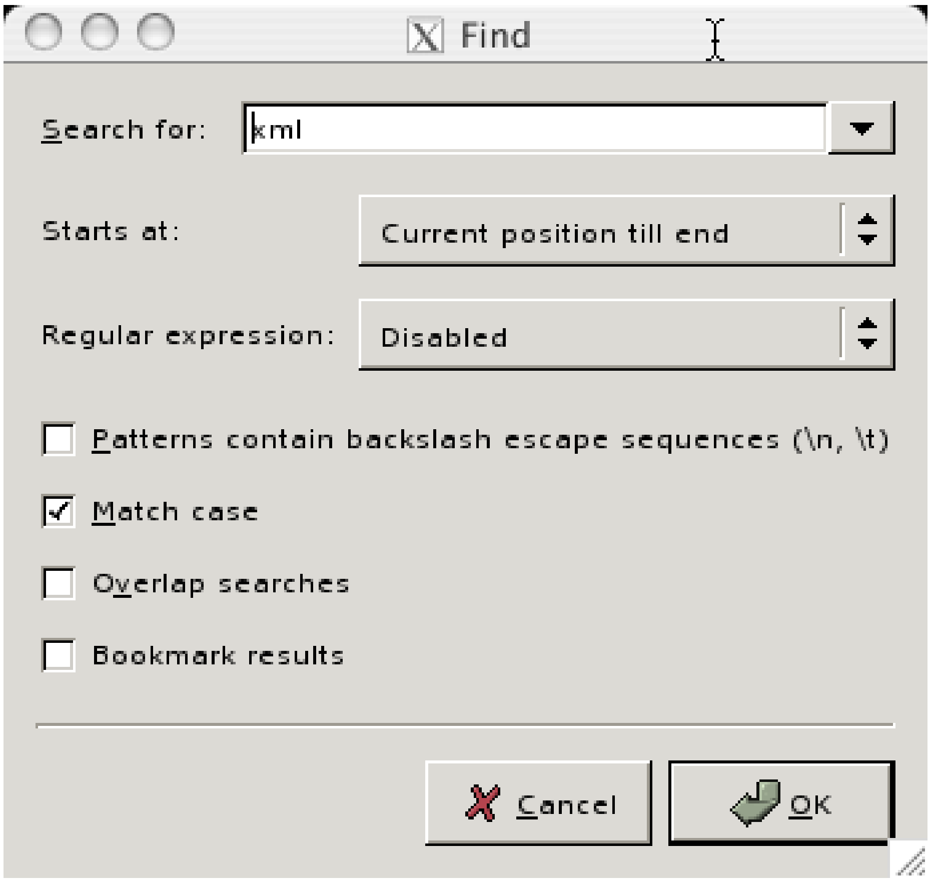
Figure 3.29. Limited search result

Notice that the search does not take into account the occurrence of the same string at line 50, since it is outside the search scope.

You can also limit the search scope to a selection range. In that case, highlight the selection before the search, and choose Beginning of selection till end of selection from the Starts at: pop up menu in the Find dialog.

Case sensitive search

By default, the search process is case insensitive. If you want to make it case sensitive, just check the Match case box in the Find dialog.



Search for: xml

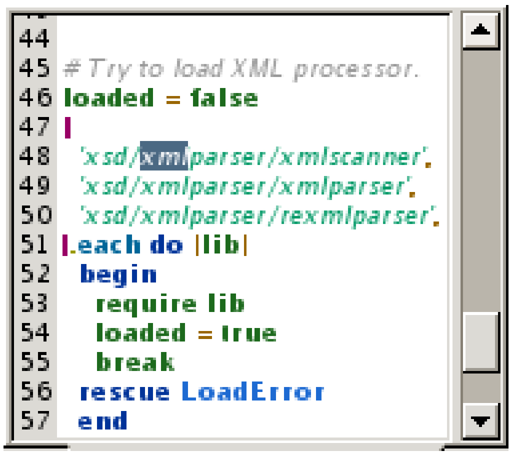
Starts at: Current position till end

Regular expression: Disabled

- [ ] Patterns contain backslash escape sequences (\n, \t)
- [✓] Match case
- [ ] Overlap searches
- [ ] Bookmark results

Figure 3.30. Making the search case sensitive

Here is the result applied to a ruby script:



```
44
45 # Try to load XML processor.
46 loaded = false
47 [
48   'xsd/xmlparser/xmlscanner',
49   'xsd/xmlparser/xmlparser',
50   'xsd/xmlparser/rexmlparser',
51 ].each do |lib|
52   begin
53     require lib
54     loaded = true
55     break
56   rescue LoadError
57   end
```

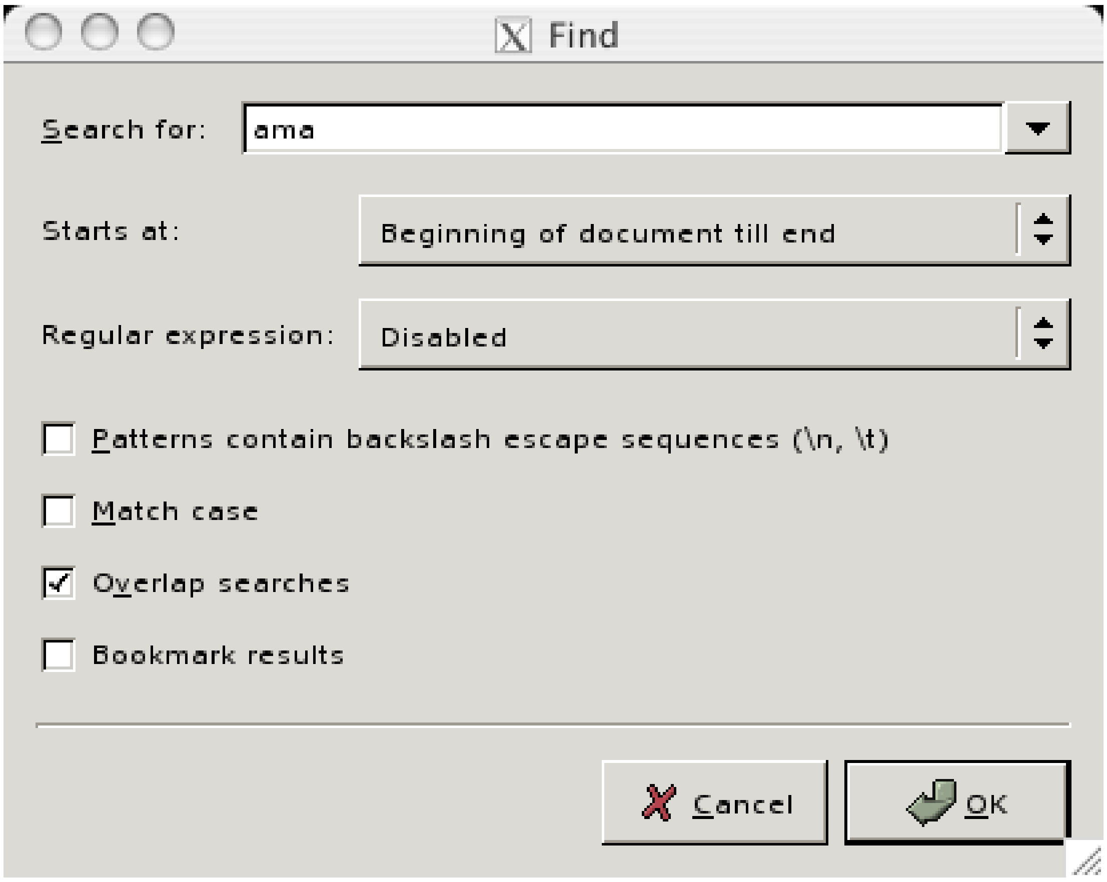
Figure 3.31. Case sensitive search result

Notice again that the result does not catch the XML string at line 45, since the search string was xml and case sensitive search was requested.

Overlapping searches

It may occur that the document contains some kind of palindrome you want to search for. The "normal" find process does not retrieve all occurrences of that kind of string.

In this case, you have to check the Overlap searches box in the Find dialog to retrieve all occurences of the string.


Find

Search for: ama

Starts at: Beginning of document till end

Regular expression: Disabled

- [ ] Patterns contain backslash escape sequences (\n, \t)
- [ ] Match case
- [x] Overlap searches
- [ ] Bookmark results

Figure 3.32. Finding overlapping strings

Applied to a shell script, the second search (with **Ctrl-F**, then **Ctrl-G**) will give the following result:

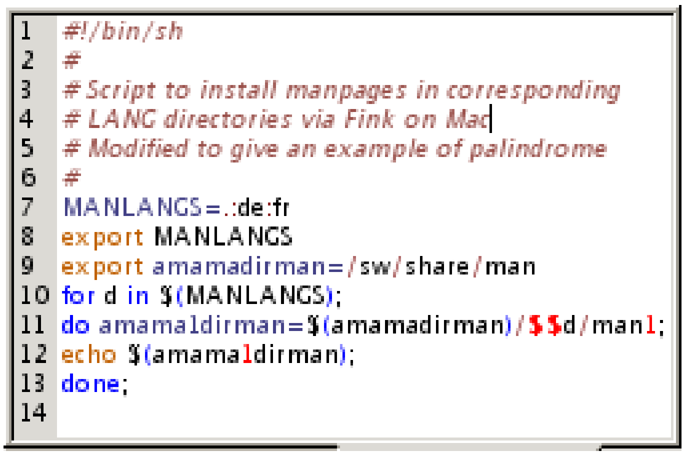

```
#!/bin/sh
#
# Script to install manpages in corresponding
# LANG directories via Fink on Mac
# Modified to give an example of palindrome
#
MANLANGS=.:de:fr
export MANLANGS
export amamadirman=/sw/share/man
for d in $(MANLANGS);
do amamaldirman=$(amamadirman)/$$d/man1;
echo $(amamaldirman);
done;

```

Figure 3.33. An overlapping string retrieved with the Find dialog

Retrieving previous search strings

Notice that the pop up menu to the right of the Search for field in the Find dialog allows you to retrieve previous search strings. They are listed in reverse order by search history, providing quicker access to the most recent searches.

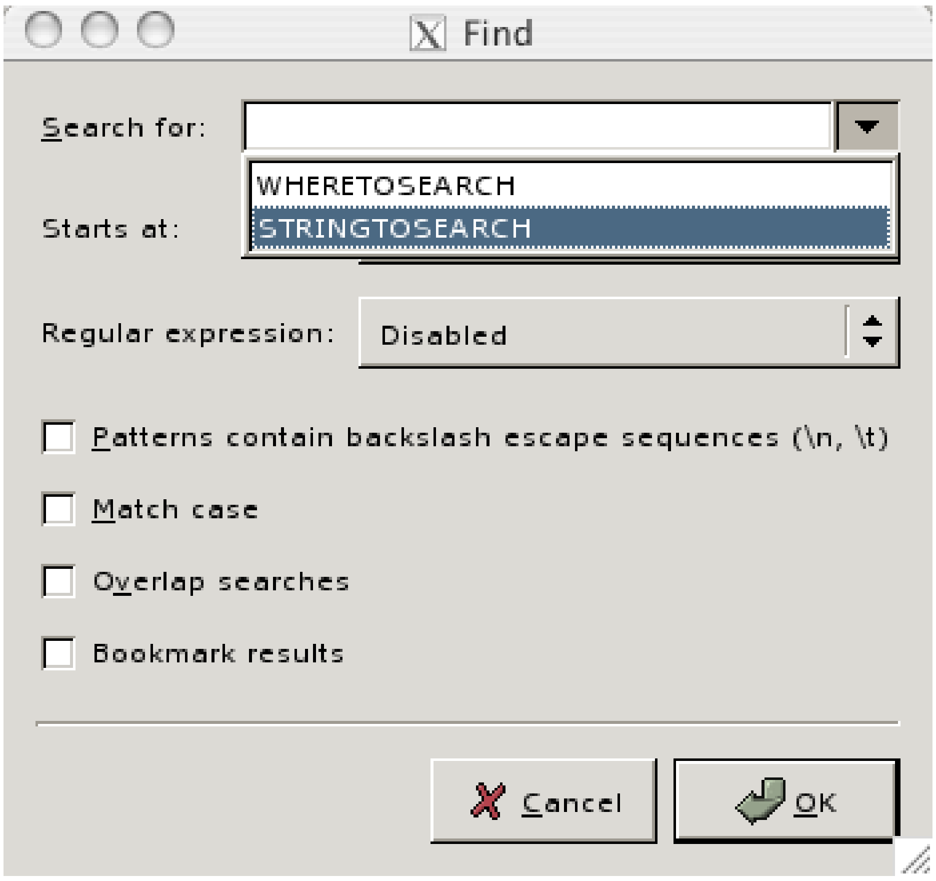


Figure 3.34. Retrieving recent searches

More on find

For an explanation of the Bookmark results box of the Find dialog, see the section called “Generating several bookmarks at once”.

You will find details on Find Again and Find from Selection in the section called “Find and Replace”.

For a quick way of switching from HTML entities to other types of encoding and changing letter cases, see the section called “Special find an replace features”.

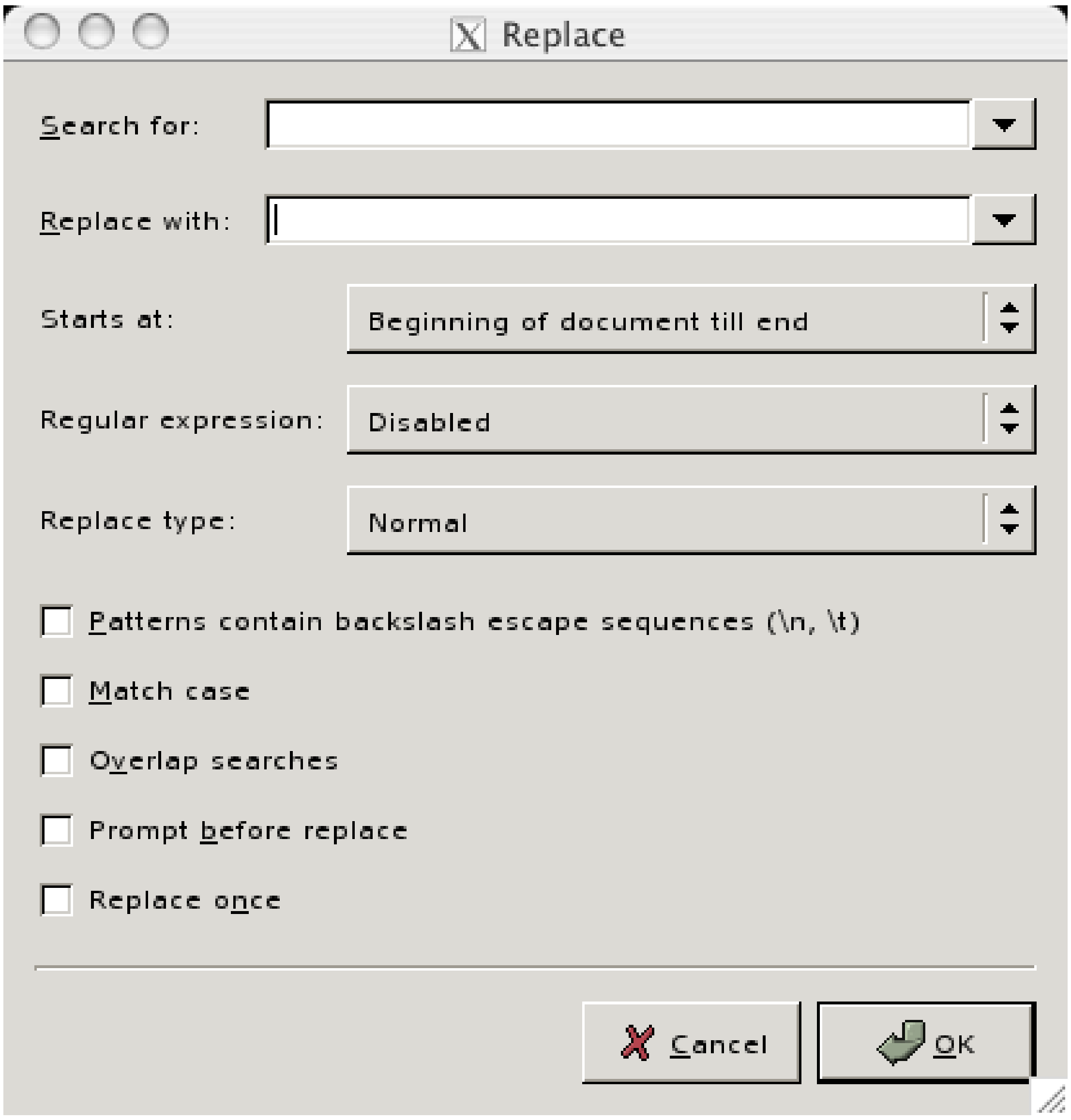
Replacing features

The Edit->Replace... (**Ctrl-H**) menu item works the same way and has all the features, the Edit->Find... (**Ctrl-F**) menu item offers.

The Replace dialog is also accessible through the contextual menu within a document.

For the features common to the Find dialog, see the section called “Searching for a word within a whole document”.

Here we will explain the features unique to the Replace dialog.



Starts at: Beginning of document till end

Regular expression: Disabled

Replace type: Normal

☐ Patterns contain backslash escape sequences (\n, \t)

☐ Match case

☐ Overlap searches

☐ Prompt before replace

☐ Replace once

Figure 3.35. The Replace dialog

Retrieving previous replace strings

As for the Search for field's pop up menu, the Replace with field's pop up menu allows you to retrieve previous strings used for replace, the most recent ones being at the top of the list.

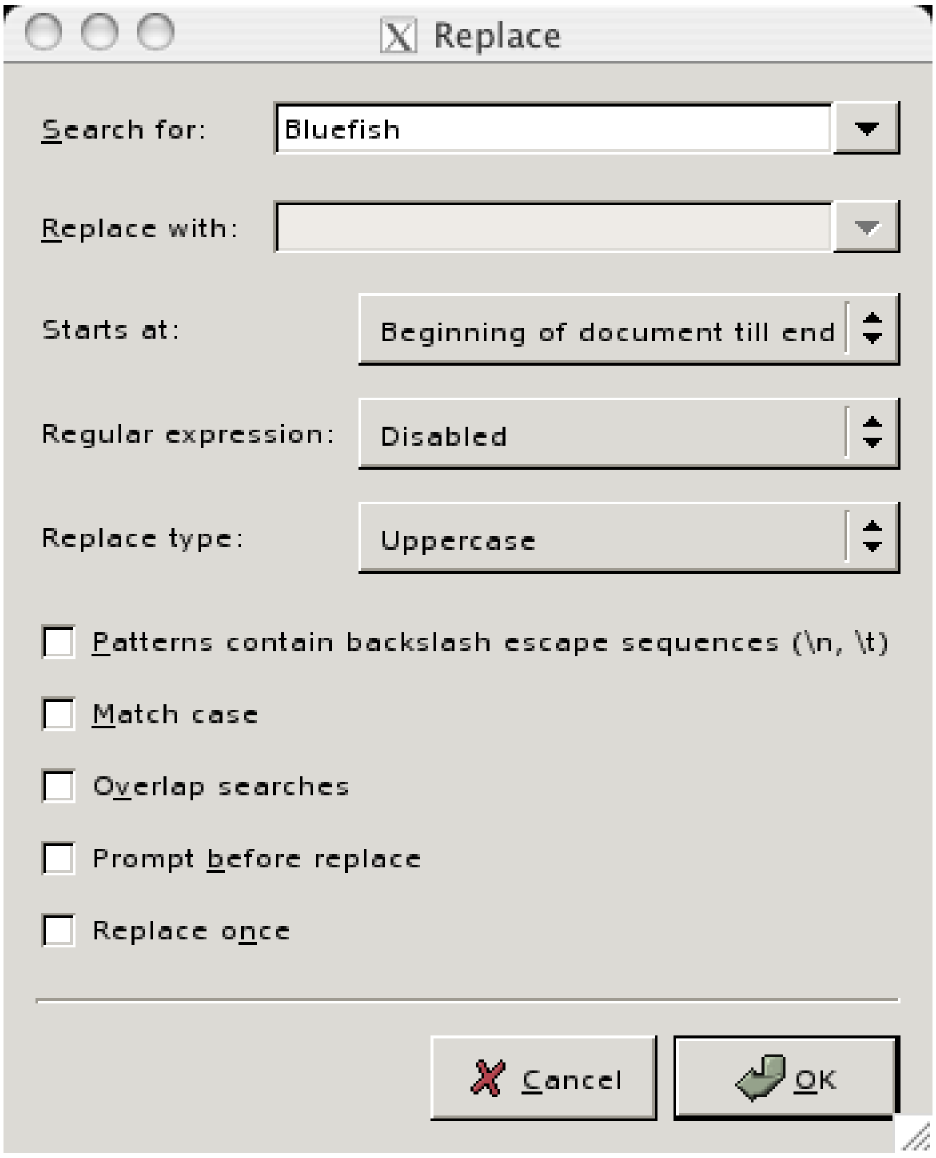
Changing letter case when replacing

If you want to change letter case when replacing, use the Replace type pop up menu.

The default choice is Normal, that is the case is not changed.

With the Uppercase replace type, the search string will be replaced with its uppercase translation.

Likewise, with the Lowercase replace type, the search string will be replaced with its lowercase translation.



Replace

Search for: Bluefish

Replace with:

Starts at: Beginning of document till end

Regular expression: Disabled

Replace type: Uppercase

☐ Patterns contain backslash escape sequences (\n, \t)

☐ Match case

☐ Overlap searches

☐ Prompt before replace

☐ Replace once

Figure 3.36. Changing letter case when replacing

Notice that in this case, the Replace with field is deactivated, thus not taken into account even if you have entered some string in it.

Choosing strings to replace

It may occur that you do not want to replace all search strings retrieved by the search process, but only some of them. In this case, check the Prompt before replace box. A Confirm replace dialog will appear for each retrieved string where you can choose to Skip this string, i.e. leave it as it is, Replace it, Replace all strings within the search scope, or Close the dialog, i.e. cancel the process.

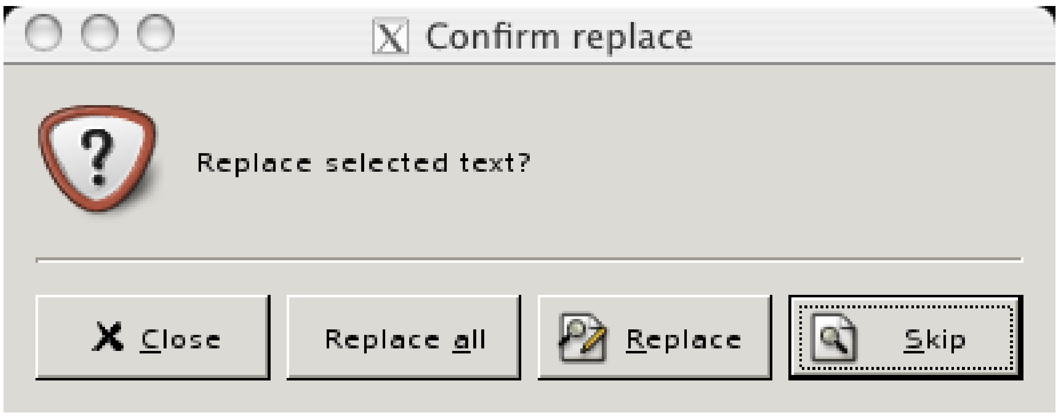


Figure 3.37. The Replace confirm dialog

If you want to replace only the first occurrence of a search string, check the Replace once box instead.

More on replace

For further explanation on replace features within Bluefish, see the section called “Find and Replace”.

File types

Different file types can change the behaviour of Bluefish. File types are recognized by their extension, or by the beginning of the file's contents. The current document type is shown in the far right of the status bar. If the type of a file is not properly detected, you can change the type using the Document->Document Type menu. See the section called “Customising Bluefish” to change these extensions.

Syntax highlighting

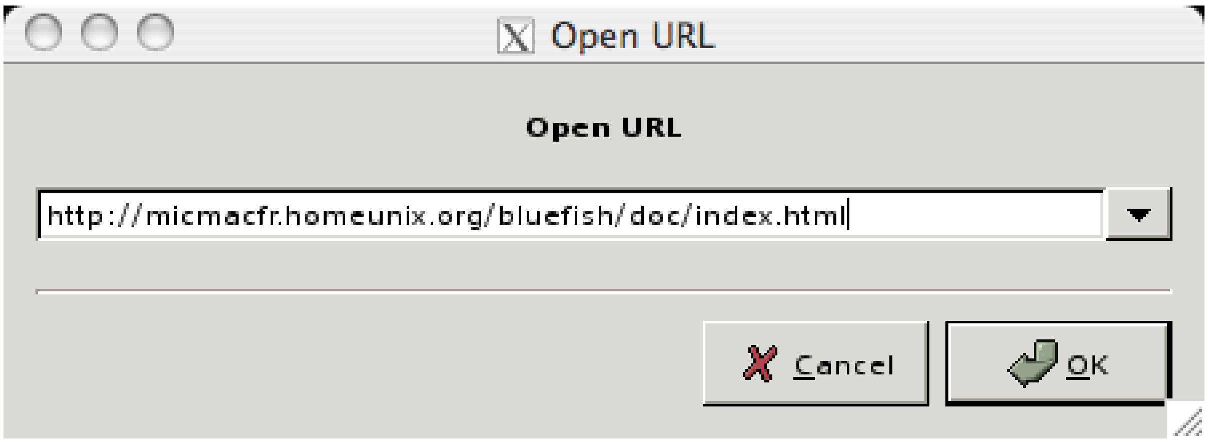
Syntax highlighting is the coloring of words that have special meaning for a language. The patterns can vary: for example, "<title>" means "start of title" in HTML, "function" means "start of function" in PHP.

While editing, Bluefish will only update the highlighting patterns in the block of text around the cursor. The number of lines (the size) of this block can be adjusted in the preferences under Editor. The syntax highlighting for the total document can be refreshed using the Document->Update Highlighting (**F5**) menu. It can be disabled in the preferences under Editor. For more information about adding or modifying syntax highlighting for existent or new languages, see here.

More on files

Remote files

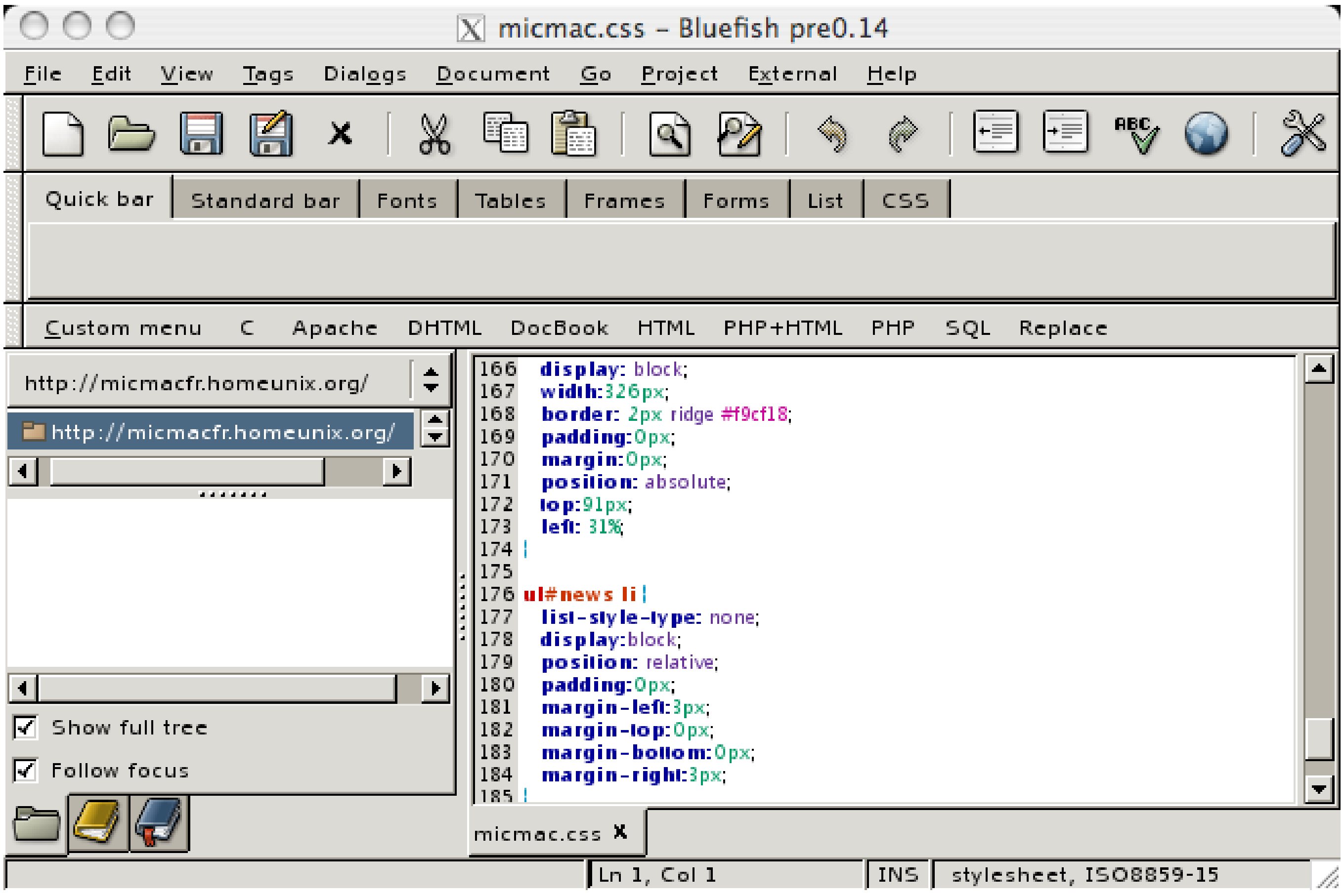
Assuming a working Internet connection, files can also be opened from the web using File->Open URL. This feature depends on your gnome_vfs setup. If it is installed and working, http://, sftp://, smb:// and possibly more types of remote services should be supported by Bluefish. Depending on your gnome_vfs version, some of these protocols are not yet fully stable, which can crash Bluefish!



http://micmacfr.homeunix.org/bluefish/doc/index.html

Figure 3.38. Opening an URL from the web

Here you can see the style sheet of an Apache web site, nicely highlighted after its opening via the Bluefish File->Open URL menu.



```
  display: block;
  width:326px;
  border: 2px ridge #f9cf18;
  padding:0px;
  margin:0px;
  position: absolute;
  top:91px;
  left: 31%;
}

ul#news li {
  list-style-type: none;
  display:block;
  position: relative;
  padding:0px;
  margin-left:3px;
  margin-top:0px;
  margin-bottom:0px;
  margin-right:3px;
}
```

Figure 3.39. A style sheet opened via the Open URL menu

Character encoding

There are many different standards for character encoding of text files. Most well known is the ASCII standard, which describes only 127 characters, and is supported by every text editor in the world. The most common standard nowadays is UTF-8, which describes thousands of characters, and is backwards compatible with ASCII.

Internally, Bluefish will always work with UTF-8. When opening a file, Bluefish has to detect the correct encoding for the file. For HTML files, the encoding should be present in a <meta name="encoding"> tag. Bluefish will always use this tag if it is available in the file. If this tag has an encoding that is not present in the Bluefish config file, this encoding is automatically added to the Bluefish config file.

The locale also defines a default encoding. If you are using a locale (a local setting, defining language, time format, currency format, number formatting etc.), Bluefish will try to load the file using the encoding defined in the locale.

Bluefish itself also has a setting for a default encoding. This is the next encoding Bluefish will try. This is also the encoding Bluefish will use for files created by Bluefish (UTF-8 by default).

If these steps fail, Bluefish will simply try every encoding defined in the Bluefish config file.

Filenames on disk can also contain non ASCII characters. All GNOME and GTK programs (including Bluefish) assume that filenames are in UTF-8 encoding. If you have filenames in the encoding of your locale on your disk, you have to set **`G_BROKEN_FILENAMES=1`** in the environment to make GNOME and GTK programs detect this encoding.

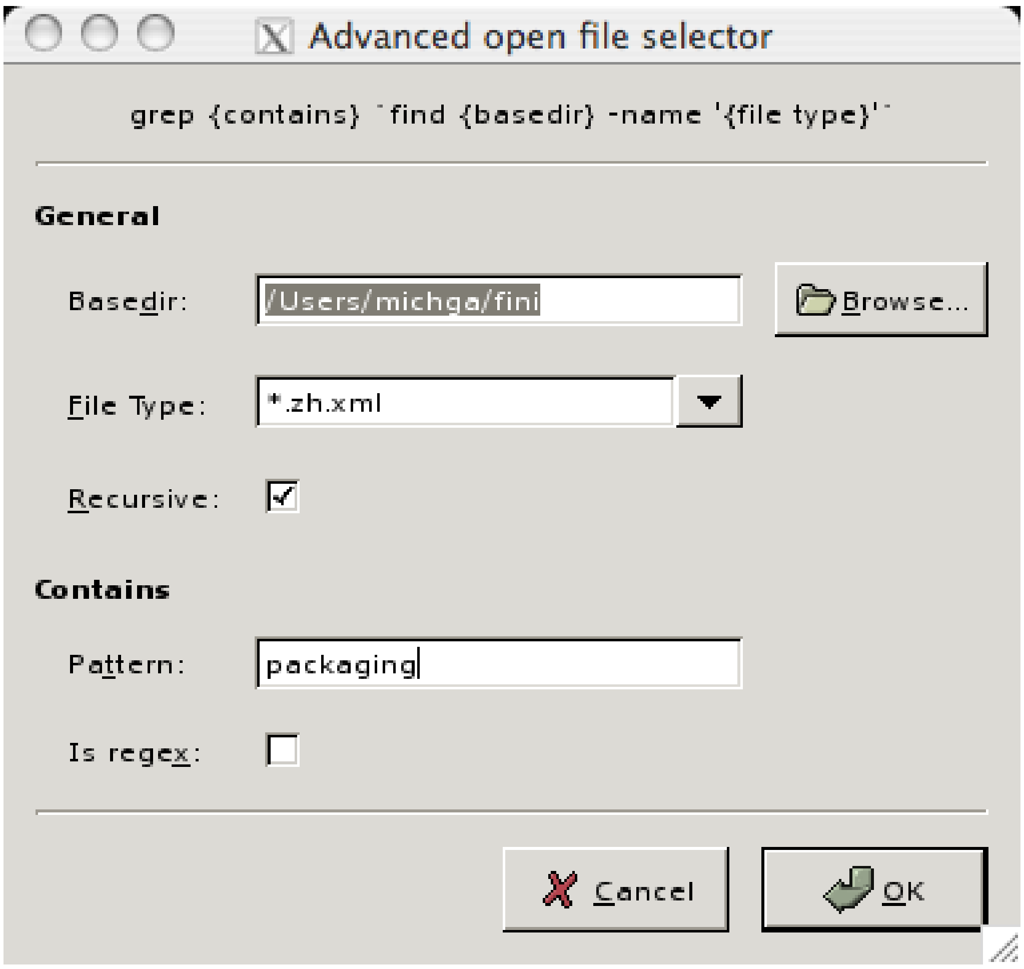
For information about writing documents in 16-bits encoded languages, such as Japanese, see the section called “Input methods”.

Open advanced

You can open multiple files at once with the File->Open Advanced... (**Shift-Ctrl-O**) menu item from a directory based on their extension or their contents. The same functionality is available from the file browser in the side panel by right-clicking a directory. This feature is available only when the find and grep utilities are installed on your system.

To open all files by extension, enter the extension in the dialog, and leave the search pattern empty. Check the recursive option if you want to include all subdirectories in the search. To open files by content, leave the extension at *, and enter a search pattern in the dialog. You can use regular expression patterns if you check the Is regex option.

You may also combine both methods. Here we open recursively all Chinese XML files in a given tree, whose contents contain the word packaging.



```
grep {contains} `find {basedir} -name '{file type}'`
```

General

Basedir: /Users/michga/fini

File Type: *.zh.xml

Recursive: ✓

Contains

Pattern: packaging

Is regex: ☐

Figure 3.40. Using the Open Advanced dialog

Navigation and Managing documents

Navigating through a document

The editing area is a standard GTK editing area. This means there are many keyboard shortcuts available to navigate through the text.

- **Ctrl-Right-Arrow** will jump to the next word boundary
- **Ctrl-Left-Arrow** will jump to the previous word boundary
- **End** will jump to the end of line
- **Home** will jump to the beginning of the line
- **Page-Up** will jump one page up
- **Page-Down** will jump one page down
- **Ctrl-Home** will jump to the top of the document
- **Ctrl-End** will jump to the end of the document

These shortcuts are also available when selecting text. Some examples:

- To select the current line, press **Home**, hold **Shift** and press **End**.
- To select the current word, press **Ctrl-Left-Arrow**, hold **Shift** and press **Ctrl-Right-Arrow**.

Navigating through many documents

Navigating through a large list of documents can be difficult. But if you right-click the document notebook tabs, you get a list of all opened documents.

Navigation between documents can also be done using the Go menu, or its shortcuts.

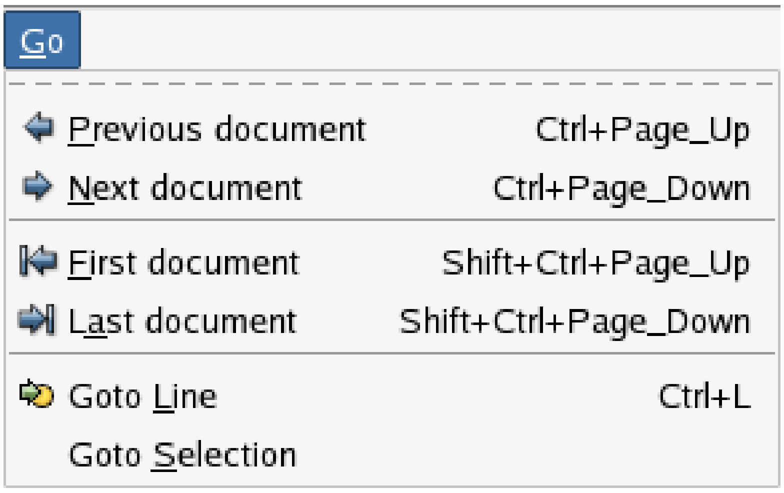

Go
Previous document Ctrl+Page_Up
Next document Ctrl+Page_Down
First document Shift+Ctrl+Page_Up
Last document Shift+Ctrl+Page_Down
Goto Line Ctrl+L
Goto Selection

Figure 3.41. Bluefish Go Menu

The shortcuts are the following:

- **Ctrl-Page-Up** will change to the previous document
- **Ctrl-Page-Down** will change to the next document
- **Shift-Ctrl-Page-Up** will change to the first document
- **Shift-Ctrl-Page-Down** will change to the last document

The Go->Goto Line (**Ctrl-L**) offers an interesting feature.

If there is some number in the document, you may select it, then click the From selection label in the Goto line dialog. Bluefish will fill in the Line number field with that number and go directly to it. The same feature is available from the Go->Goto Selection.

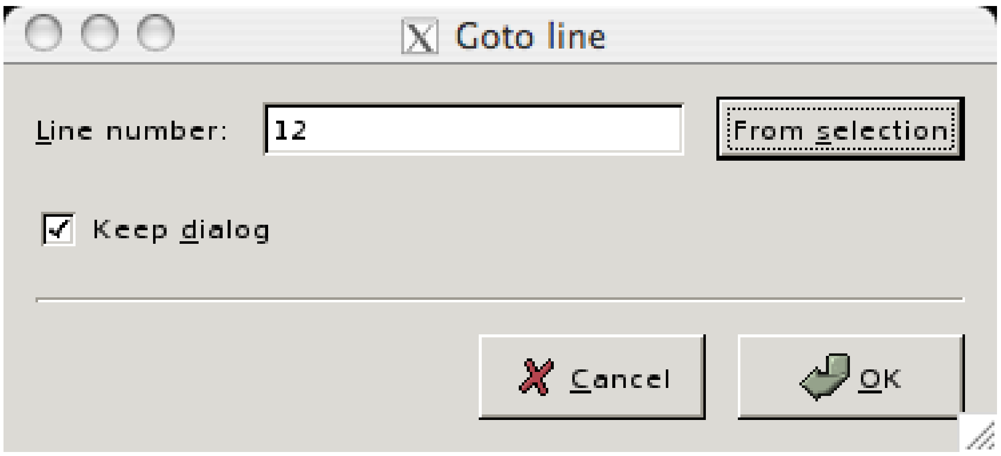


Figure 3.42. Using the Goto Line dialog

Tip

Check the Keep dialog box to keep the dialog open, when you plan to access several parts of the document by line numbers.

Projects

The projects are a sort of *saved state* of Bluefish. Thus, they are a very convenient way to work with files scattered all over your disks or to pick up only the files you are interested in within a huge tree. Projects features are accessible through the Project menu.

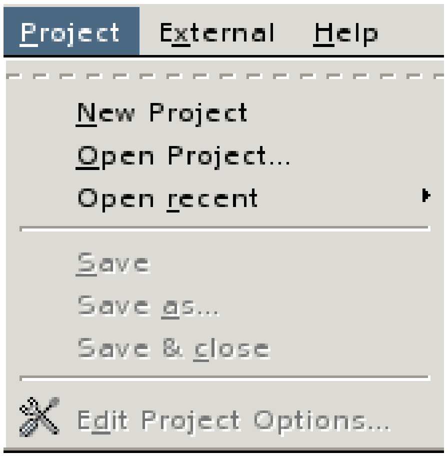
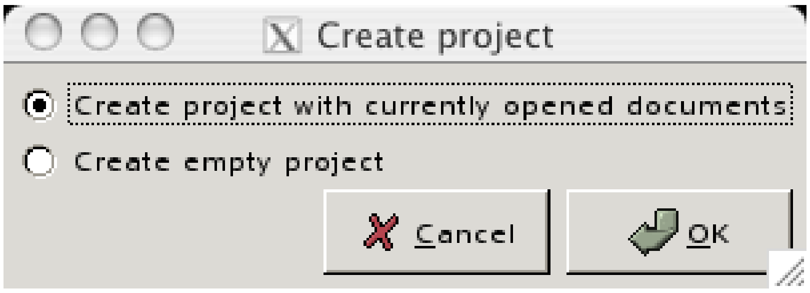


Figure 3.43. The Bluefish Project Menu

Procedure 3.3. Creating a New Project

1. Click on the Project->New Project

   If some documents are already opened, check the appropriated box in the Create project dialog.

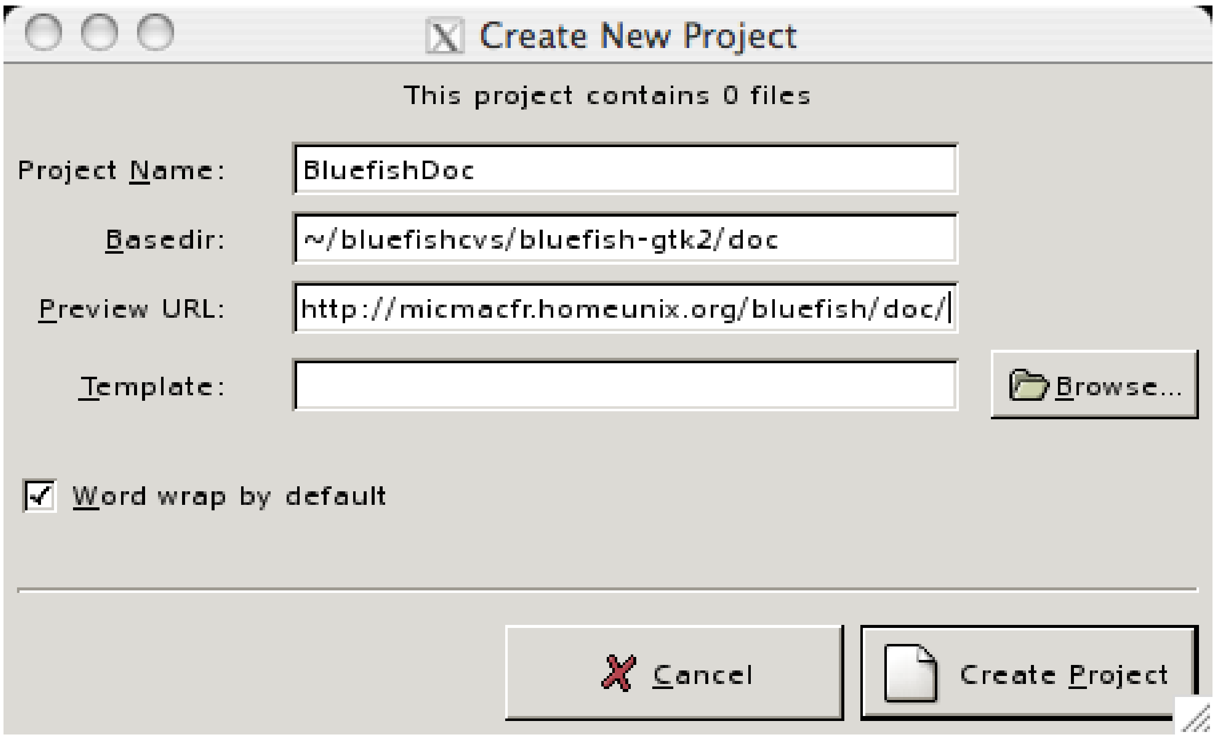
   Create project

   [● Create project with currently opened documents]

   ○ Create empty project

   Figure 3.44. The Create Project dialog

2. Fill in the fields in the Create New Project dialog

   Create New Project

   This project contains 0 files

   Project Name: BluefishDoc

   Basedir: ~/bluefishcvs/bluefish-gtk2/doc

   Preview URL: http://micmacfr.homeunix.org/bluefish/doc/

   Template:

   [✓] Word wrap by default

   Figure 3.45. Creating a New Project

With a Basedir the file browser in the side panel shows only the files within its hierarchy. With gnome-vfs support, the Basedir can be remote, as `smb://user:pass@server/someshare/` or `sftp://someserver/somedir`.

The Preview URL allows Bluefish to launch the browser to the appropriate URL, for example `http://localhost/Bluefish`. This can be very convenient for testing server side scripting languages like PHP, JSP, etc.

If the Template field is used, Bluefish will use the template file's contents for new files, which can be requested either via the New button on the main tool bar or File->New (**CtrlN**). Otherwise an empty document will be created.

3. Once the project is created, you need to tell Bluefish where you want to save it. An Enter Bluefish project filename dialog will be displayed. Notice that you can save the project in a location different from the files to which the project points.

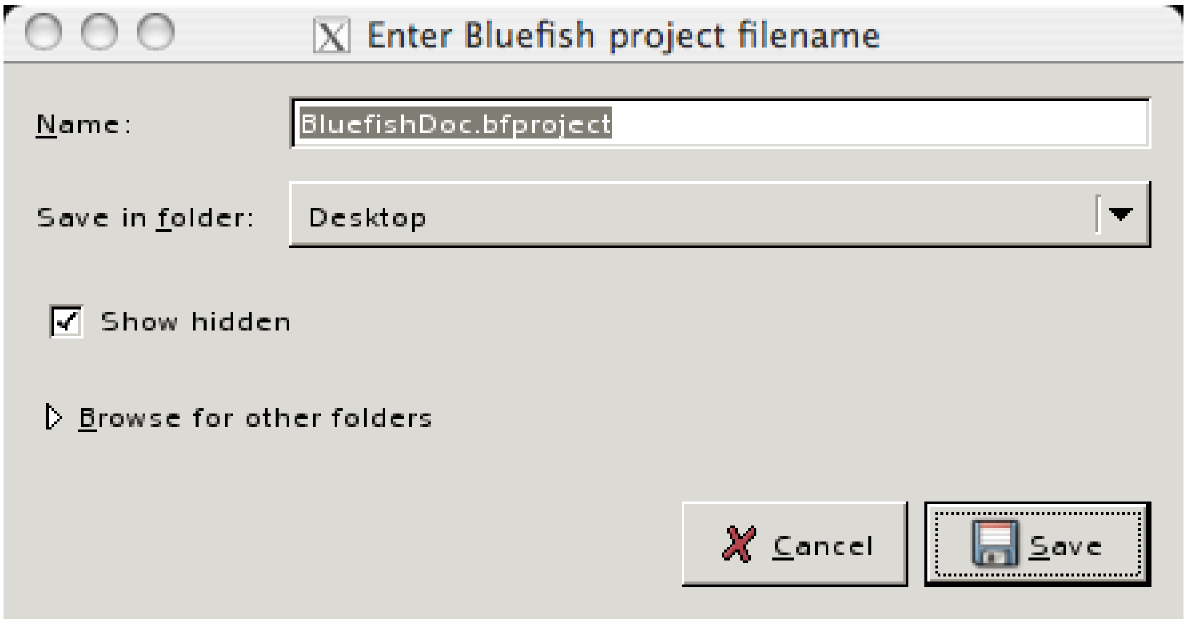


Figure 3.46. Entering Bluefish Project Filename

To open a project, you have the choice between Project->Open Project... or Project->Open recent. When you choose the former, a Selecting a Bluefish Project dialog is presented to you.

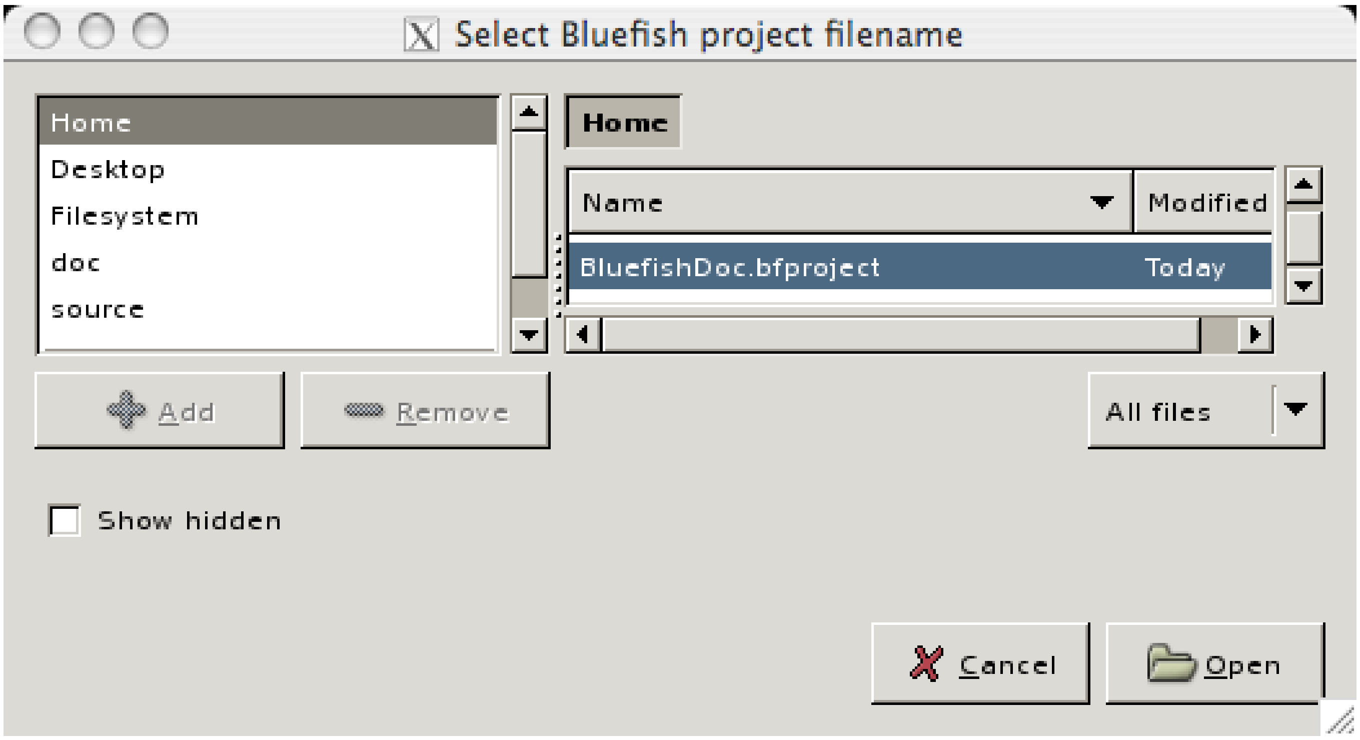
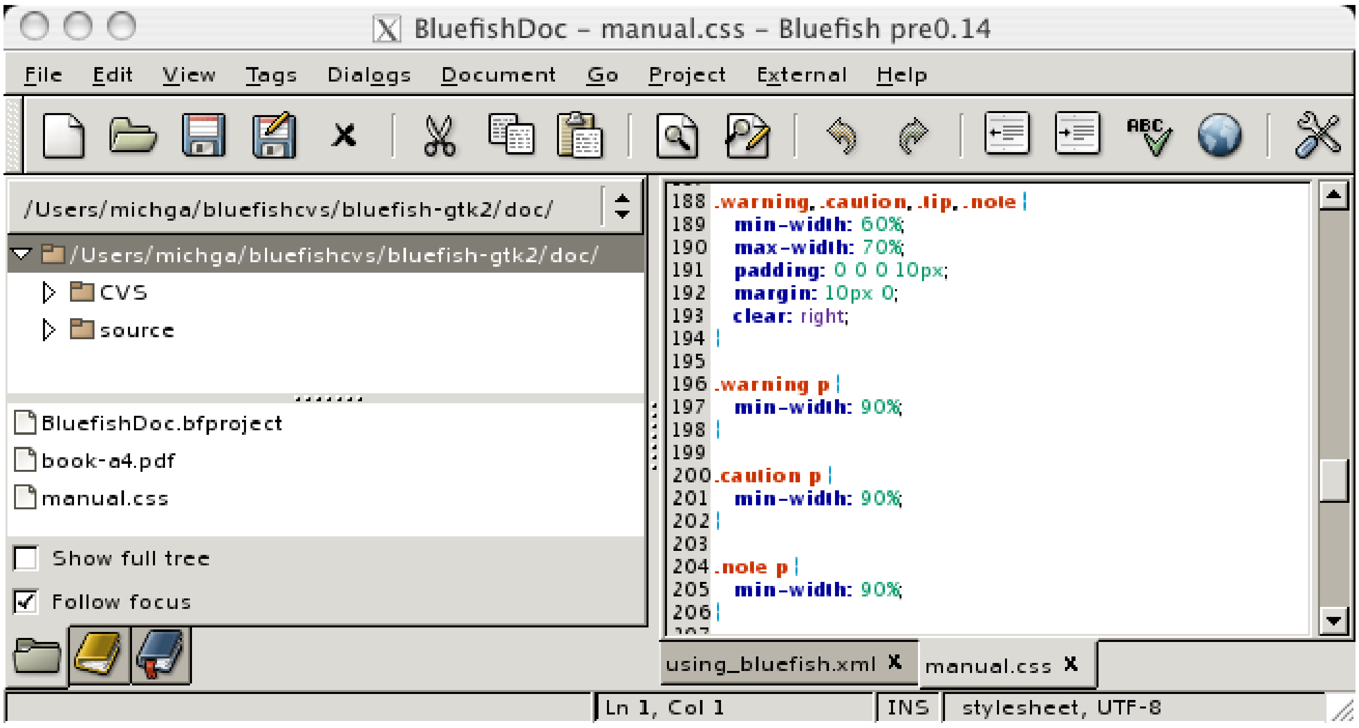


Figure 3.47. Selecting a Bluefish Project

To save the project under its current name/location, use Project->Save or Project->Save & close; to save it under a new name/location, use Project->Save as.... If any file in the project has changed, a dialog will allow you to save the file, discard the changes, or cancel. All files open when the project is saved are automatically opened the next time you open the project.



```
.warning, .caution, .tip, .note {
  min-width: 60%;
  max-width: 70%;
  padding: 0 0 0 10px;
  margin: 10px 0;
  clear: right;
}

.warning p {
  min-width: 90%;
}

.caution p {
  min-width: 90%;
}

.note p {
  min-width: 90%;
}
```

Figure 3.48. Opening a Bluefish Project

Notice that the side panel only shows the tree related to the project.

Also, the recently used files in that project are shown in the File->Open recent menu item.

A project also saves some basic Bluefish settings, giving the project its own customized Bluefish setup. Currently, the word wrap preference and the state of various tool and menu bars are saved in a project file. The project file itself is simply a text file in the standard Bluefish format (same format as the config file). This format is `key: value`. Here is an example:

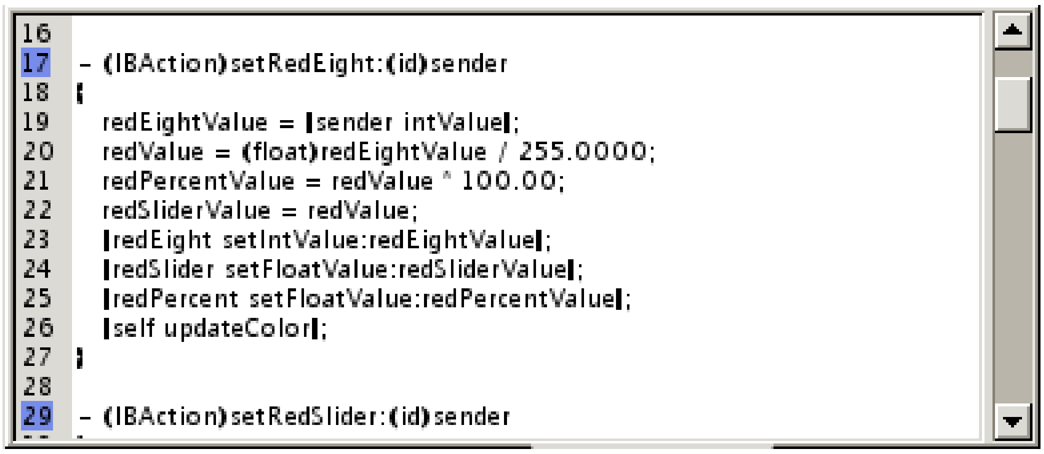
```
name: BluefishDoc
basedir: ~/bluefishcvs/bluefish-gtk2/doc/
webdir: http://micmacfr.homeunix.org/bluefish/doc
template:
view_main_toolbar: 1
view_left_panel: 1
view_custom_menu: 1
view_html_toolbar: 1
word_wrap: 1
```

Bookmarks

In Bluefish you can add bookmarks to a line in the text, and you can later use the bookmark to quickly jump to this location, or even to open the document referred to by the bookmark at that line.

Bookmarks can be added to the current cursor location by using the Edit->Add Bookmark (**Ctrl-D**) menu item; or by right-clicking in the text, and selecting Add bookmark. You can delete a bookmark using the Delete bookmark item in the document contextual menu.

Each bookmark in a given document is marked by a blue background in the line number margin.



```

- (IBAction)setRedEight:(id)sender
{
   redEightValue = [sender intValue];
   redValue = (float)redEightValue / 255.0000;
   redPercentValue = redValue * 100.00;
   redSliderValue = redValue;
   [redEight setIntValue:redEightValue];
   [redSlider setFloatValue:redSliderValue];
   [redPercent setFloatValue:redPercentValue];
   [self updateColor];
}

- (IBAction)setRedSlider:(id)sender
```

Figure 3.49. How bookmarks are marked

Bookmarks can be temporary or permanent. Permanent bookmarks are stored, and temporary bookmarks are gone after Bluefish is closed. The default is set in the preferences under Editor.

Bookmarks can be found in the third tab of the side panel, sorted by document and line number.

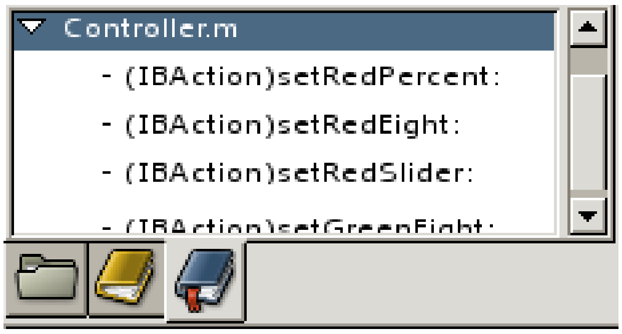


Figure 3.50. Bookmarks in the side panel

If you right click a bookmark in the bookmark tab of the side panel, you get a pop up menu with several options.

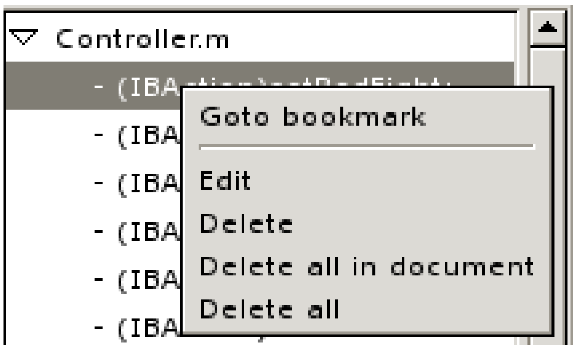
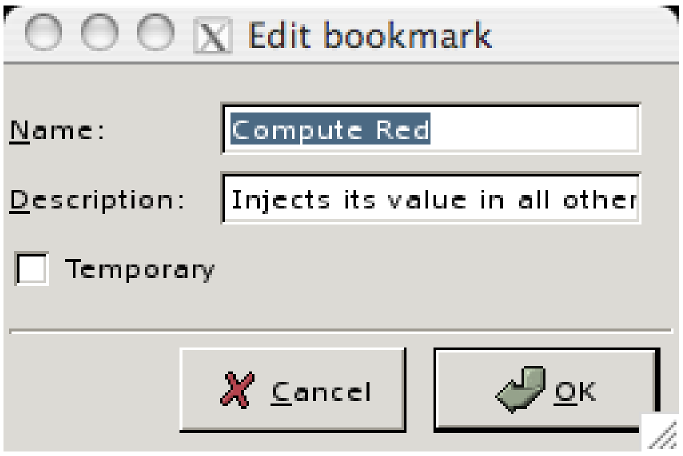


Figure 3.51. Contextual menu on bookmark in the side panel

The Goto bookmark item allows you to go to the bookmark location in the document, opening it if needed.

The Edit item allows you to change a bookmark from temporary to permanent or the other way around, to name it, and to give it a short description.

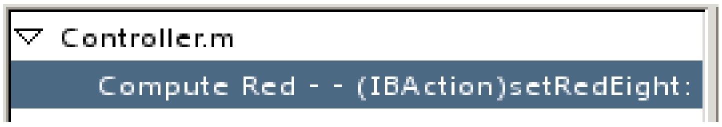


Name: Compute Red

Description: Injects its value in all other

Figure 3.52. Editing a bookmark

Note that after naming a bookmark, the default name - first characters of the bookmarked line - is displayed after the new name.



Controller.m

Compute Red - - (IBAction)setRedEight:

Figure 3.53. A named bookmark

Via this contextual menu, you may also delete a bookmark, delete all bookmarks in the active document, or delete all bookmarks stored in the bookmark tab of the side panel. The latter ones are also available when you right click the name of a document in this tab.

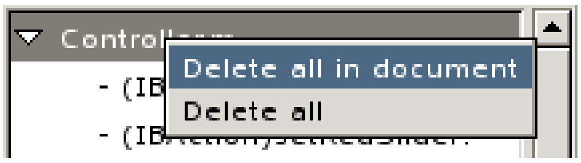
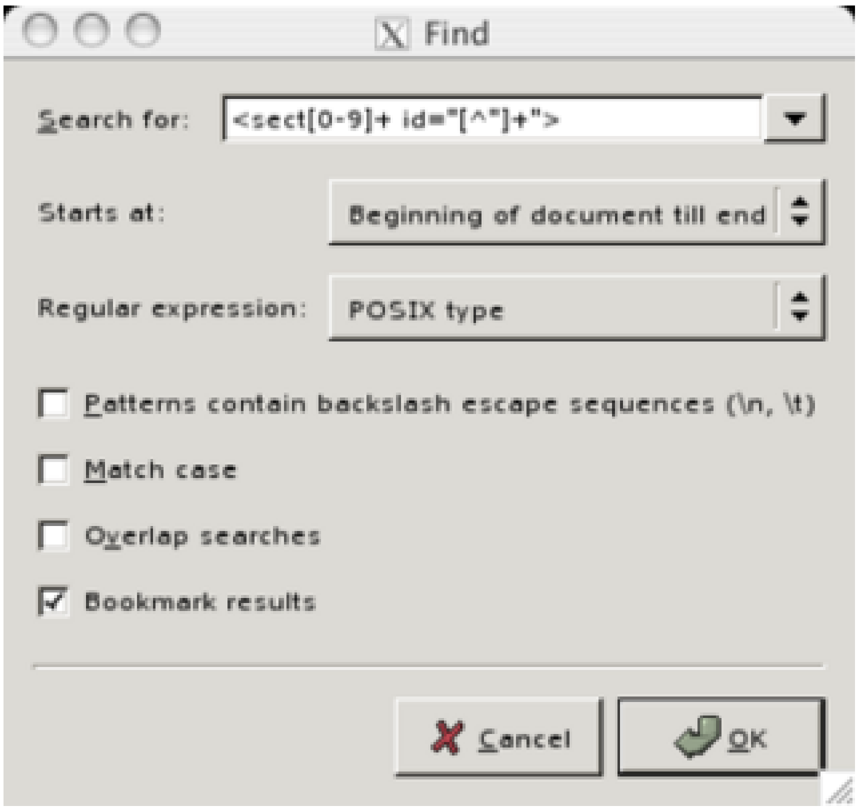


Figure 3.54. The contextual menu on a document in the bookmark tab

Generating several bookmarks at once

To add many bookmarks at once, use the Edit->Find (**Ctrl-F**) dialog. Check the Bookmark result option, and all search results will be added to your bookmarks.

For example, the XML files for this manual have sections, each identified by a header like `<sect1 id="nameofthesection">`. A way to automatically get a bookmark to every section is to search for the following posix regular expression pattern: `<sect[0-9]+ id="[^"]+">` and bookmark all results.



Find

Search for: <sect[0-9]+ id="[^"]+">

Starts at: Beginning of document till end

Regular expression: POSIX type

☐ Patterns contain backslash escape sequences (\n, \t)

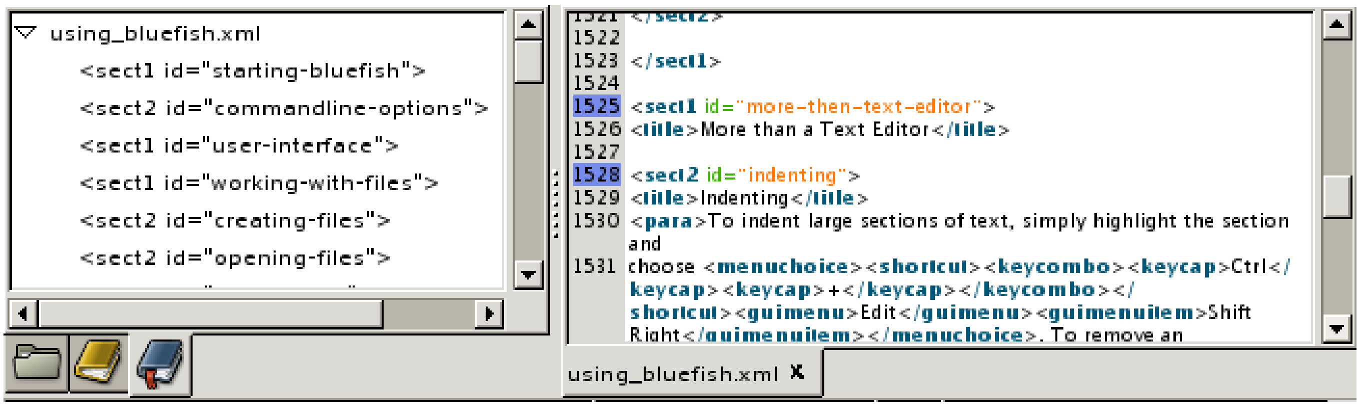
☐ Match case

☐ Overlap searches

☑ Bookmark results

Figure 3.55. Bookmarking with Posix regular expression

Here is the result:

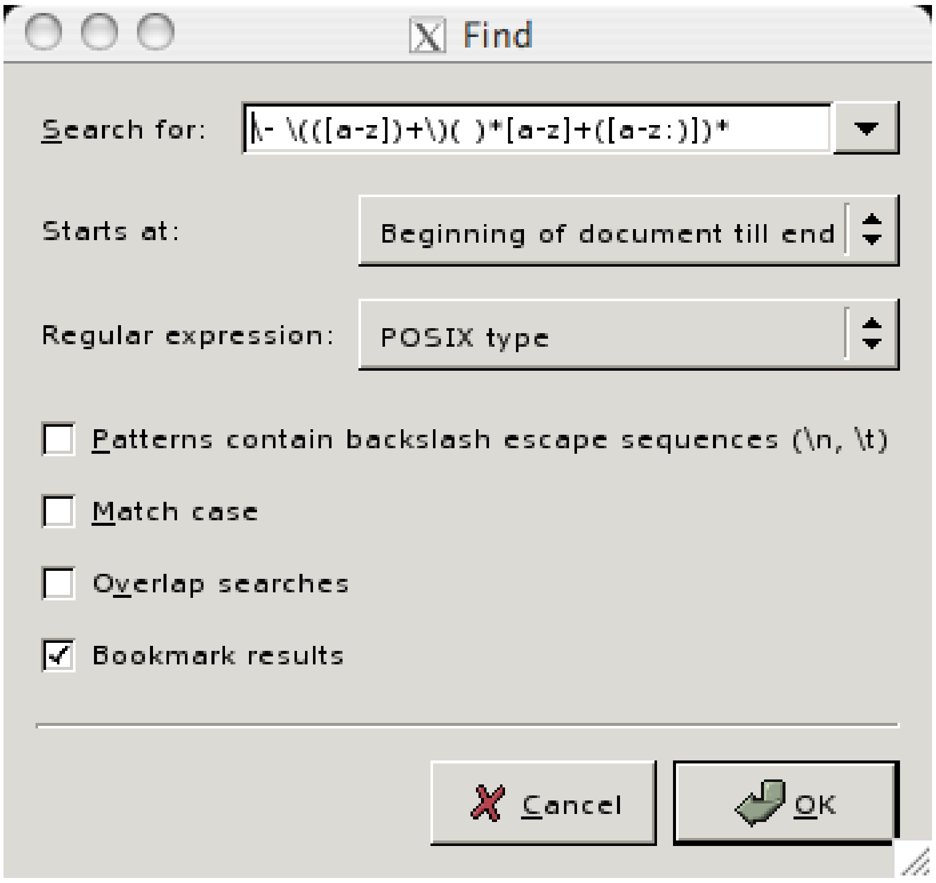


```
using_bluefish.xml
    <sect1 id="starting-bluefish">
    <sect2 id="commandline-options">
    <sect1 id="user-interface">
    <sect1 id="working-with-files">
    <sect2 id="creating-files">
    <sect2 id="opening-files">
```

```
</sect2>

</sect1>

<sect1 id="more-then-text-editor">
<title>More than a Text Editor</title>

<sect2 id="indenting">
<title>Indenting</title>
<para>To indent large sections of text, simply highlight the section
and
choose <menuchoice><shortcut><keycombo><keycap>Ctrl</
keycap><keycap>+</keycap></keycombo></
shortcut><guimenu>Edit</guimenu><guimenuitem>Shift
Right</guimenuitem></menuchoice>. To remove an
```

Figure 3.56. Bookmarks with Posix regular expression

Here are two examples which bookmarks all functions in Objective-C and PHP files with POSIX or PERL regular expressions:



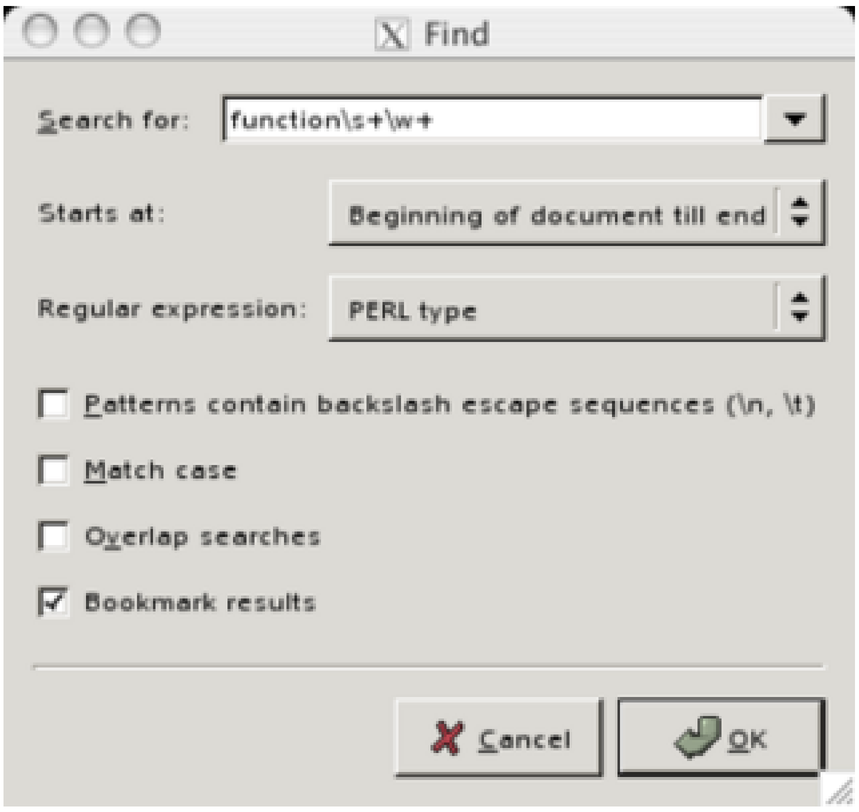
Search for: `\- \(([a-z])+\)( )*[a-z]+([a-z:)])*`

Starts at: Beginning of document till end

Regular expression: POSIX type

- [ ] Patterns contain backslash escape sequences (\n, \t)
- [ ] Match case
- [ ] Overlap searches
- [✓] Bookmark results

Figure 3.57. Bookmarking Objective C functions via the Find menu



Search for: `function\s+\w+`

Starts at: Beginning of document till end

Regular expression: PERL type

- [ ] Patterns contain backslash escape sequences (\n, \t)
- [ ] Match case
- [ ] Overlap searches
- [✓] Bookmark results

Figure 3.58. Bookmarking PHP functions via the Find menu

Check the section called “Find and Replace Using Regular Expressions” for more information on finding and replacing with regular expression in Bluefish.

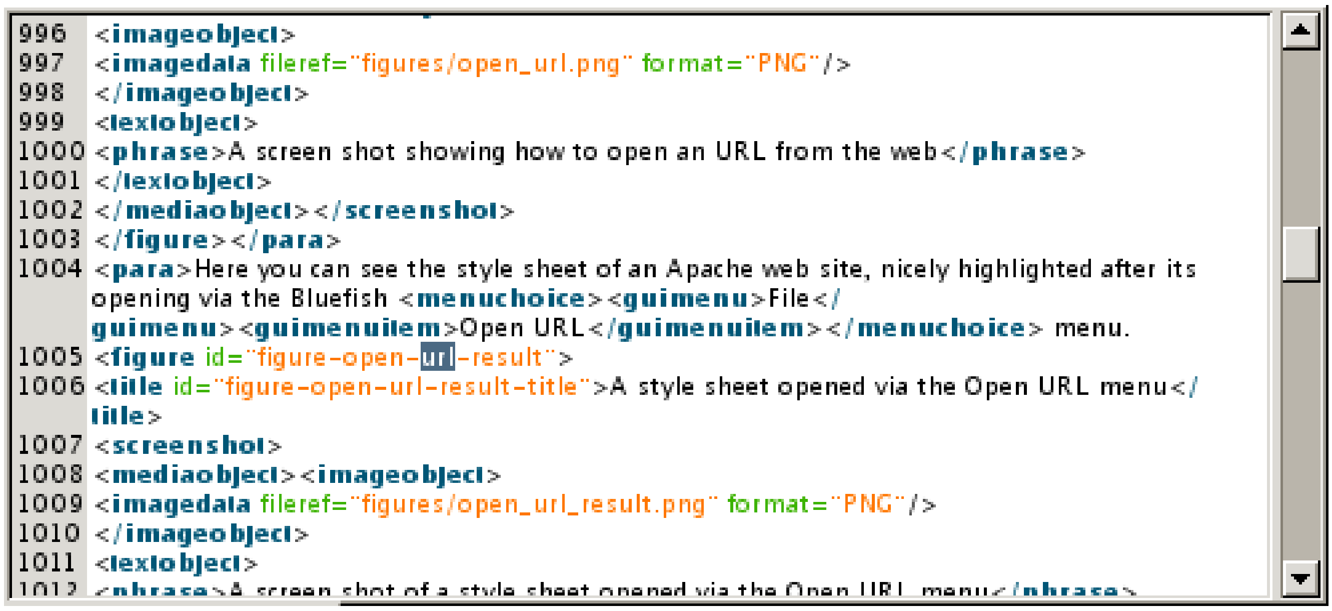
Find and Replace

The Edit->Find (**Ctrl-F**) and Edit->Replace... (**Ctrl-H**) menu items will simply start their corresponding dialogs described in the section called “Basic Find and Replace”.

Find Again

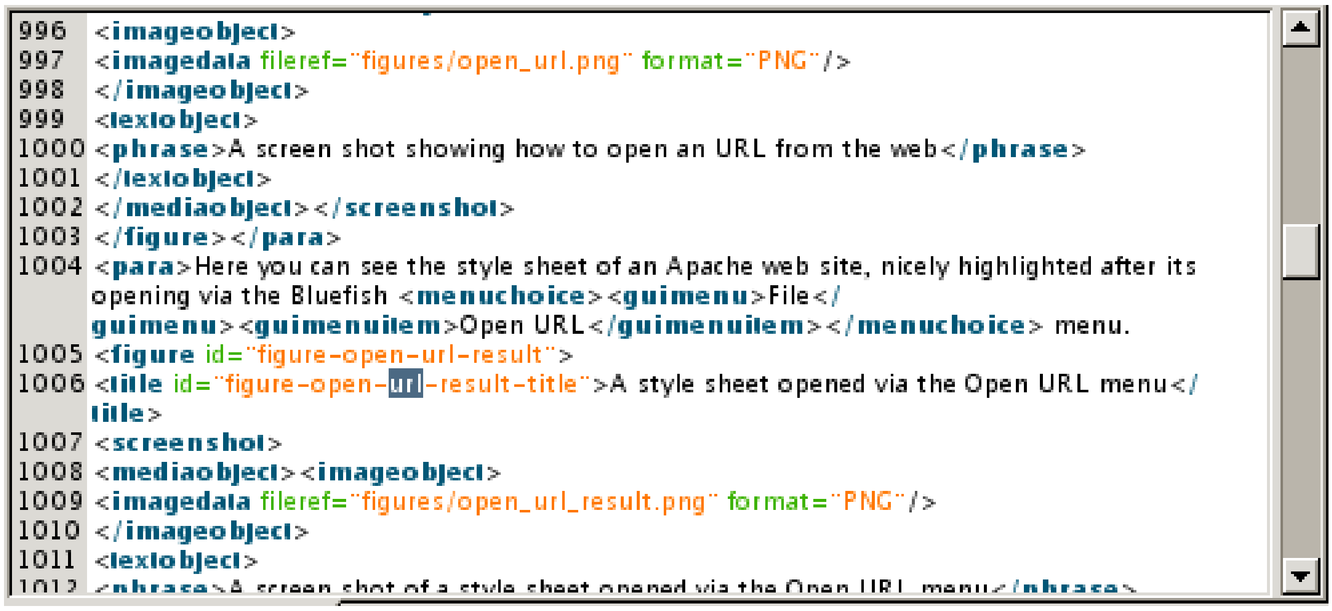
The Edit->Find again (**Ctrl-G**) menu item will repeat the last used search. It will continue the search after the position where the previous search was stopped. If the end of file is reached, it will signaled it if the search operates on a unique file, nevertheless you can continue the search from the top of file with **Ctrl-G** after dismissing the Search: no match found dialog. If the search operates on multiple files, it will continue with the next file.

Here you can see how the Find again process operates on two successive searches for the word "url" in an xml file:



```
<imageobject>
<imagedata fileref="figures/open_url.png" format="PNG"/>
</imageobject>
<textobject>
<phrase>A screen shot showing how to open an URL from the web</phrase>
</textobject>
</mediaobject></screenshot>
</figure></para>
<para>Here you can see the style sheet of an Apache web site, nicely highlighted after its opening via the Bluefish <menuchoice><guimenu>File</guimenu><guimenuitem>Open URL</guimenuitem></menuchoice> menu.
<figure id="figure-open-url-result">
<title id="figure-open-url-result-title">A style sheet opened via the Open URL menu</title>
<screenshot>
<mediaobject><imageobject>
<imagedata fileref="figures/open_url_result.png" format="PNG"/>
</imageobject>
<textobject>
<phrase>A screen shot of a style sheet opened via the Open URL menu</phrase>
```

Figure 3.59. Nth occurrence with Find Again



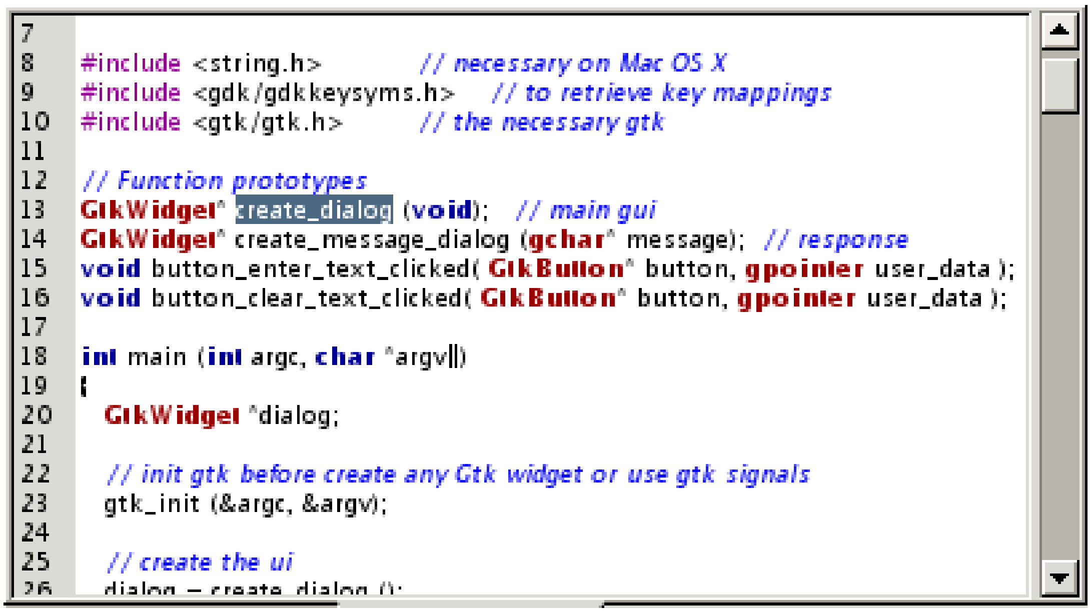
```
<imageobject>
<imagedata fileref="figures/open_url.png" format="PNG"/>
</imageobject>
<textobject>
<phrase>A screen shot showing how to open an URL from the web</phrase>
</textobject>
</mediaobject></screenshot>
</figure></para>
<para>Here you can see the style sheet of an Apache web site, nicely highlighted after its opening via the Bluefish <menuchoice><guimenu>File</guimenu><guimenuitem>Open URL</guimenuitem></menuchoice> menu.
<figure id="figure-open-url-result">
<title id="figure-open-url-result-title">A style sheet opened via the Open URL menu</title>
<screenshot>
<mediaobject><imageobject>
<imagedata fileref="figures/open_url_result.png" format="PNG"/>
</imageobject>
<textobject>
<phrase>A screen shot of a style sheet opened via the Open URL menu</phrase>
```

Figure 3.60. Nth+1 occurrence with Find Again

Find from Selection

The Edit->Find from selection menu item will search for the currently selected text. If you select for example the name of a function, in bluefish, or in any other program, and you choose find from selection Bluefish will start a new search for this selected string.

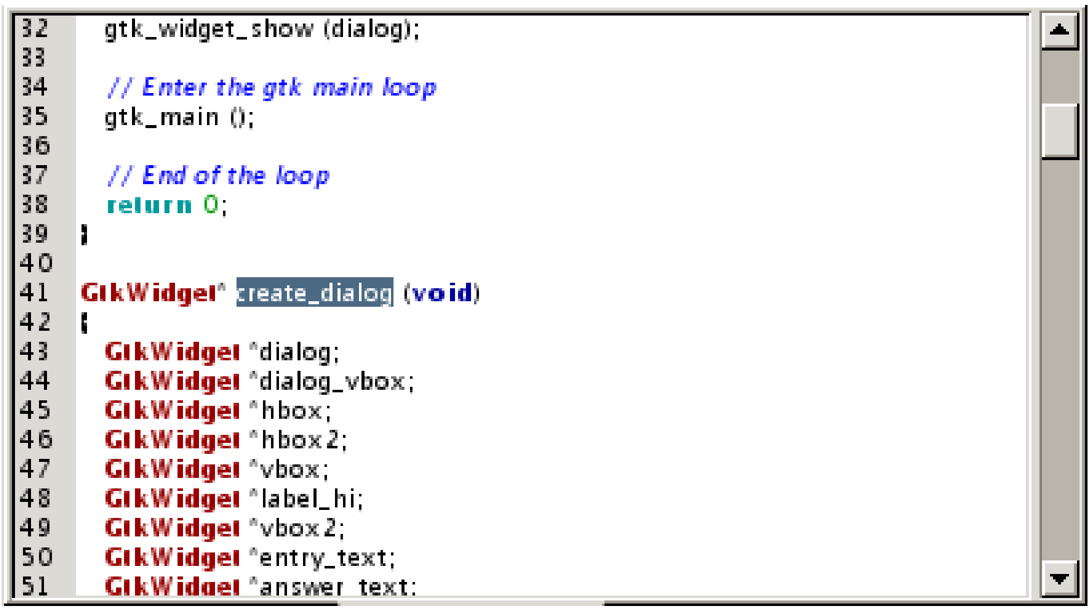
Here we have selected a function in a C file:



```

#include <string.h>         // necessary on Mac OS X
#include <gdk/gdkkeysyms.h>    // to retrieve key mappings
#include <gtk/gtk.h>        // the necessary gtk

// Function prototypes
GtkWidget* create_dialog (void);   // main gui
GtkWidget* create_message_dialog (gchar* message);  // response
void button_enter_text_clicked( GtkButton* button, gpointer user_data );
void button_clear_text_clicked( GtkButton* button, gpointer user_data );

int main (int argc, char *argv[])
{
  GtkWidget *dialog;

  // init gtk before create any Gtk widget or use gtk signals
  gtk_init (&argc, &argv);

  // create the ui
  dialog = create_dialog ();
```

Figure 3.61. Selecting a string for subsequent search

Clicking on Edit->Find from selection gives the following result:



```
  gtk_widget_show (dialog);

  // Enter the gtk main loop
  gtk_main ();

  // End of the loop
  return 0;
}

GtkWidget* create_dialog (void)
{
  GtkWidget *dialog;
  GtkWidget *dialog_vbox;
  GtkWidget *hbox;
  GtkWidget *hbox2;
  GtkWidget *vbox;
  GtkWidget *label_hi;
  GtkWidget *vbox2;
  GtkWidget *entry_text;
  GtkWidget *answer_text;
```

Figure 3.62. Finding a string from selection

Next occurrences of the string can be found with **Ctrl-G** as usual.

Find and Replace Using Regular Expressions

With find and replace you can do incredibly powerful searches. We have already seen some of them in the section called “Generating several bookmarks at once”, which deserve some explanation here.

Example 3.1. Retrieving all sections in an xml book

The regular POSIX expression `<sect[0-9]+ id="[^"]+">` can be split into:

- `<sect`: a string beginning by `<sect`
- `[0-9]+ ` : followed by one or more (the `+` part) characters in the range of 0 to 9 (the `[0-9]` part), i.e. digits, followed by a space
- `id="`: followed by the string `id`, followed by an equal sign, followed by a double-quote
- `[^"]+`: followed by one or more (the `+` part) not double-quote characters (the `[^"]` part - `^` is a not)
- `">`: followed by a double-quote, and ending with a > sign

Therefore, it matches any string of type `<sectn id="nameofthesection">`, where n is a positive integer.

Example 3.2. Retrieving all functions in an Objective C file

In a simplified example, an objective C function may have two forms:

1. ` - (IBAction)nameofthefunction:(id)parameter`
2. ` - (void) nameofthefunction`

We will try to make a pattern from those forms:

- Hyphens and parentheses have special meanings in regular expressions, hence we need to escape them, i.e. to put before each of them a backslash, so that they will be interpreted as normal characters.

  Thus, `- (` is matched by: `\- \(`
- `IBAction` or `void` are a non empty sequence of alphabetical characters. We have already seen something similar in the previous example.

  They are matched by: `[a-z]+`, that is one or more characters in the range of a to z.
- Another parenthesis matched by: `\)`.
- A space or no space at all, it is matched by: ` *`, that is a space followed by an asterisk, which means 0 or more times the preceding character.
- A non empty sequence of characters, matched by `[a-z]+` as already seen.
- A colon or no colon at all, which is matched by: `[:]*`.

Thus the whole POSIX regular expression is: `\- \([a-z]+\) *[a-z]+[:]*`. In the example, we have grouped the parts with parentheses, you may prefer this simplified form, though it is not recommended.

Example 3.3. Retrieving all functions in a PHP file

A php function has the form `function nameofthefunction(listofparameters)`, where the list of parameters can be empty. To match it with a PERL regular expression, you have to know that `\s matches any white space` and `\w` matches any alphanumerical character as well as white spaces.

Thus, the matching regular expression is: `function\s+\w+`.

Now, if you want to capture also the function's parameters, you have to add:

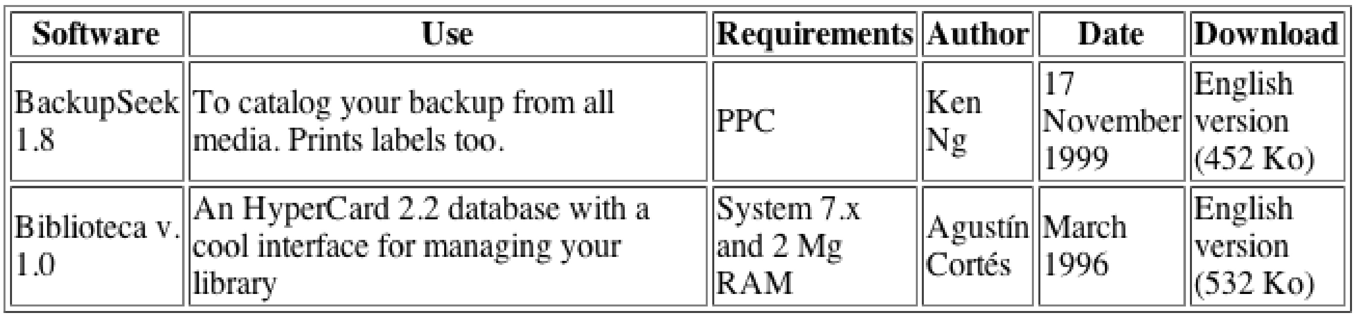
- An opening parenthesis: `\(`. Remember parentheses should be escaped with a backslash.
- Zero or more characters, none of them being a closing parenthesis: `[^\)]*`
- A closing parenthesis: \)

The PERL regular expression becomes: `function\s+\w+\([^\)]*\)`.

Here is a new example which transforms a table into a definition list inside an html file.

Example 3.4. Transforming a table into a definition list

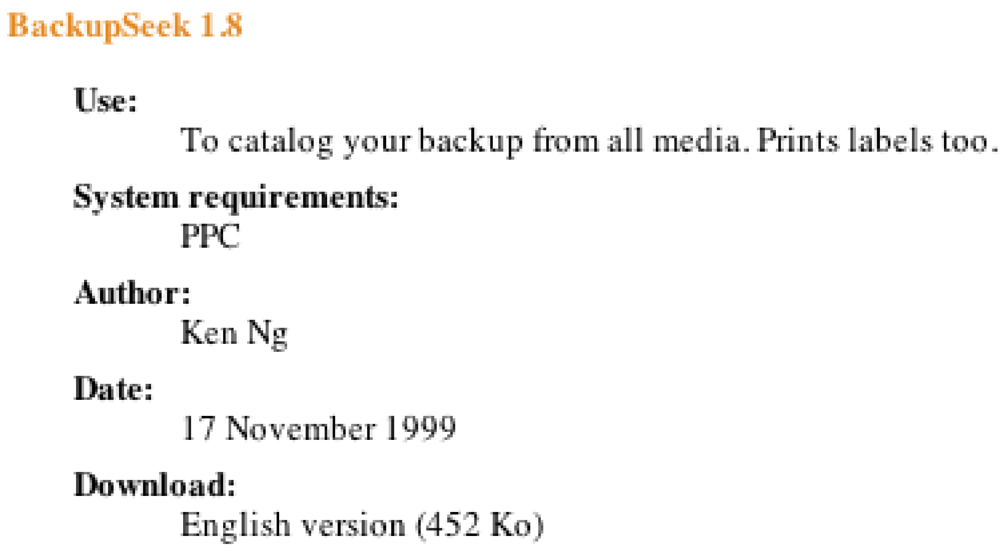
Say you have the following table:



Software	Use	Requirements	Author	Date	Download
BackupSeek 1.8	To catalog your backup from all media. Prints labels too.	PPC	Ken Ng	17 November 1999	English version (452 Ko)
Biblioteca v. 1.0	An HyperCard 2.2 database with a cool interface for managing your library	System 7.x and 2 Mg RAM	Agustín Cortés	March 1996	English version (532 Ko)

Figure 3.63. The table before transformation

You want to transform it in the following definition list:



BackupSeek 1.8

Use:
To catalog your backup from all media. Prints labels too.

System requirements:
PPC

Author:
Ken Ng

Date:
17 November 1999

Download:
English version (452 Ko)

Figure 3.64. The table after transformation

For the rendering, you will use the following css style sheet:

```
.st2 {
    font-weight: 900;
    color: #e38922;
    margin-left: 30px;
}
dl {
    font-weight: 900;
    margin-left: 55px;
}
dt {
    margin-top: 6px;
}
.dd1 {
    font-style: normal;
    font-weight: 400;
}
```

The table's source code is the following:

```
<table border="1">
<tr>
<th>Software</th>
<th>Use</th>
<th>Requirements</th>
<th>Author</th>
<th>Date</th>
<th>Download</th>
</tr>
<tr>
<td>BackupSeek 1.8</td>
<td>To catalog your backup from all media. Prints labels too.</td>
<td>PPC</td>
<td>Ken Ng</td>
<td>17 November 1999</td>
<td>English version (452 Ko)</td>
</tr>
<tr>
<td>Biblioteca v. 1.0</td>
...
</tr>
</table>
```

The definition list's source code will be the following:

```
<p class="st2">BackupSeek 1.8</p>
<dl>
<dt>Use:</dt><dd><span class="dd1">To catalog your backup from all media. \
Prints labels too.</span></dd>
<dt>System requirements:</dt><dd><span class="dd1">PPC</span></dd>
<dt>Author:</dt><dd><span class="dd1">Ken Ng</span></dd>
<dt>Date:</dt><dd><span class="dd1">17 November 1999</span></dd>
<dt>Download:</dt><dd><span class="dd1">English version (452 Ko)</span></dd>
</dl>
<p class="st2">Biblioteca v. 1.0</p>
...
</dl>
```

Comparing both chunks of code, we see that the variable sequence of characters to capture is the one between one `<td>` tag and its closing `</td>` tag. That sequence can be interpreted as one or more characters which are not a `<`. We have already seen that. This is expressed as: `[^<]+`

To be able to retrieve it later, we need to embed it into parentheses. Thus, the string becomes: `([^<]+)`

Next, this sequence is embedded into `<td>` and `</td>`, which is expressed simply concatenating the strings: `<td>([^<]+)</td>`

We should also add the end of line character, which is expressed as: `\n`. The regular expression now describes a whole line: `<td>([^<]+)</td>\n`

As we cannot use variables to retrieve the headers of the table, we will merely repeat that string five times, so that the regular expression matches exactly the six lines of importance to us.

Note

Do not type it six times in the search field. Select the string, use the shortcuts **Ctrl-C** to copy it, move to the end of the string with the right arrow, and use **Ctrl-V** five times to paste it at the end of the string.

The regular expression becomes (backslashes are inserted at end of line just for the purpose of not to have too long lines):

```
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n
```

Those lines are at their turn embedded into `<tr>` and `</tr>` tags each of them on their own line, which can be expressed as: `<tr>\n` for the first one, and `</tr>\n` for the second one. We add those strings respectively at the beginning and at the end of our regular expression, which becomes:

```
<tr>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
<td>([^<]+)</td>\n \
</tr>\n
```

Now that we have described the search pattern, we will build the replace pattern. Each expression embedded into parentheses in the search string can be retrieved with `\x`, where x is an integer starting at 0 for the first expression, 1 for the second, etc. All others parts in the final string are fixed strings which we will express as they are.

The first line becomes (note the \n at the end to match the end of line character):

```
<p class="st2">\0</p>\n
```

The second line (again, note the \n to match the end of line characters):

```
<dl>\n<dt>Use:</dt><dd><span class="dd1">\1</dd>\n
```

And finally the whole replace pattern is:

```
<p class="st2">\0</p>\n \
<dl>\n<dt>Use:</dt><dd><span class="dd1">\1</dd>\n \
<dt>System requirements:</dt><dd><span class="dd1">\2</span></dd>\n \
<dt>Author:</dt><dd><span class="dd1">\3</span></dd>\n \
<dt>Date:</dt><dd><span class="dd1">\4</dd>\n \
<dt>Download:</dt><dd><span class="dd1">\5</span></dd>\n</dl>\n
```

After entering both patterns, choose PERL type in the Regular expression drop down list, check the Patterns contain backslash escape sequences (\n, \t) and click OK.

After replacement occurred, you have to remove the table headers and the last `</table>` tag and to insert the link to the style sheet.

Note that if some lines contain a < sign, the table row will not be translated, but others will.

In the Find and Replace dialogs it is not possible to insert the keys **Enter** or **Tab**. A simple way to do it is to copy two lines in a row from the current document into the Find or Replace dialog, this way you retrieve the end of line character. The same applies for **Tab**. A more elaborated way to do it is to use escaped characters to represent these characters. A new line character, produced by pressing the **Enter** key, is represented as `\n`. Use `\t` for a tab. To get an actual backslash, just escape the backslash, `\\`. There are many other escape characters used in regular expressions.

Note

To enable the escaped characters in your searches check the Patterns contain backslash sequences (\n, \t) option.

If you have any search and replace patterns you use often, you can also add them to the Custom Menu. Check the section called “Custom menu” for more information.

For more information about regular expressions you might want to read man 1 perlre, man 3 pcrepattern, man 3 regex or man 7 regex, or read any of the great Internet sites about regular expressions. As you become more familiar with regular expressions, you will realize that they make Bluefish a very powerful editor.

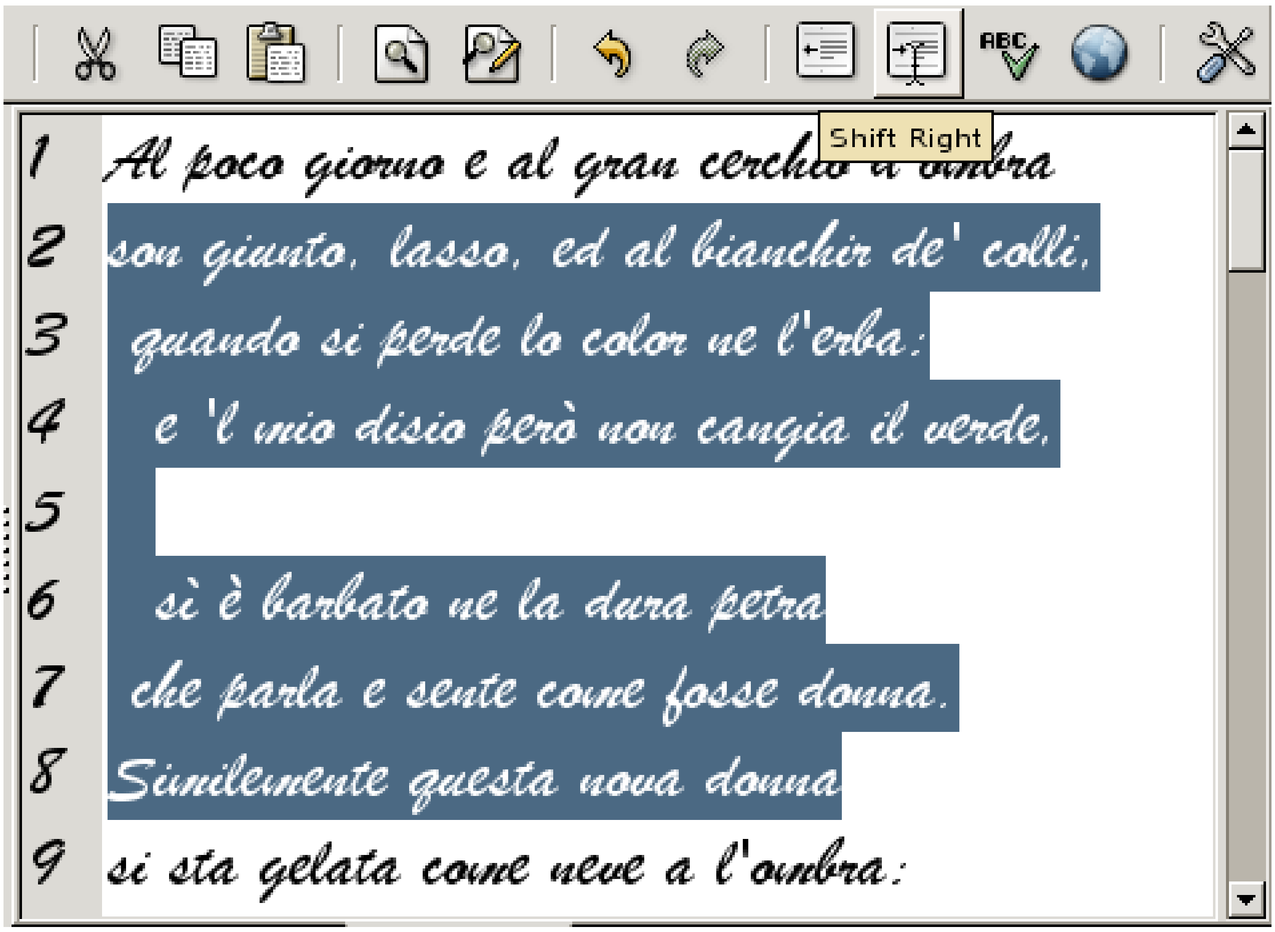
More than a Text Editor

Indenting

To indent large sections of text, simply highlight the section and choose Edit->Shift Right (**Ctrl-.**). To remove an indentation, choose Edit->Shift Left (**Ctrl-,**). There are corresponding buttons in the tool bar for these menu options (see later in this text).

By default, Bluefish will use tabs for indenting, but can be configured to use spaces if you have Use spaces to indent, not tabs selected in the Editor preferences panel. The number of spaces used is the same as the Tab width option in the same preferences panel.

Here's an extract of Dante's work indented with the Shift Right button in the main tool bar:



```
Al poco giorno e al gran cerchio d'ombra
son giunto, lasso, ed al bianchir de' colli,
 quando si perde lo color ne l'erba:
  e 'l mio disio però non cangia il verde,

  sì è barbato ne la dura petra
 che parla e sente come fosse donna.
Similemente questa nova donna
si sta gelata come neve a l'ombra:
```

Figure 3.65. Indenting part of a text

Auto tag closing

By default, Bluefish will automatically produce closing tags for HTML and XML documents. For example, if you type `<p>` within an HTML document, bluefish will produce `</p>`. So, as soon as you finish typing a non-empty HTML tag, meaning the tag is supposed to have a closing tag, Bluefish will help you out and close the tag automatically. This feature can be turned off by unchecking the Document->Auto Close HTML Tags (**Ctrl-T**) menu option.

Bluefish has two modes for tag closing, an XML mode and an HTML mode. In XML mode, Bluefish will add a closing tag to any tag that is not closed itself with />. In HTML mode, Bluefish excludes all known tags that do not need a closing tag, such as <br> and <img>.

Bluefish will choose the mode based on the file type of the document. In the filetype preferences panel, the default mode for each file type can be set. See the section called “Modifying file types” for more info.

Spell checker

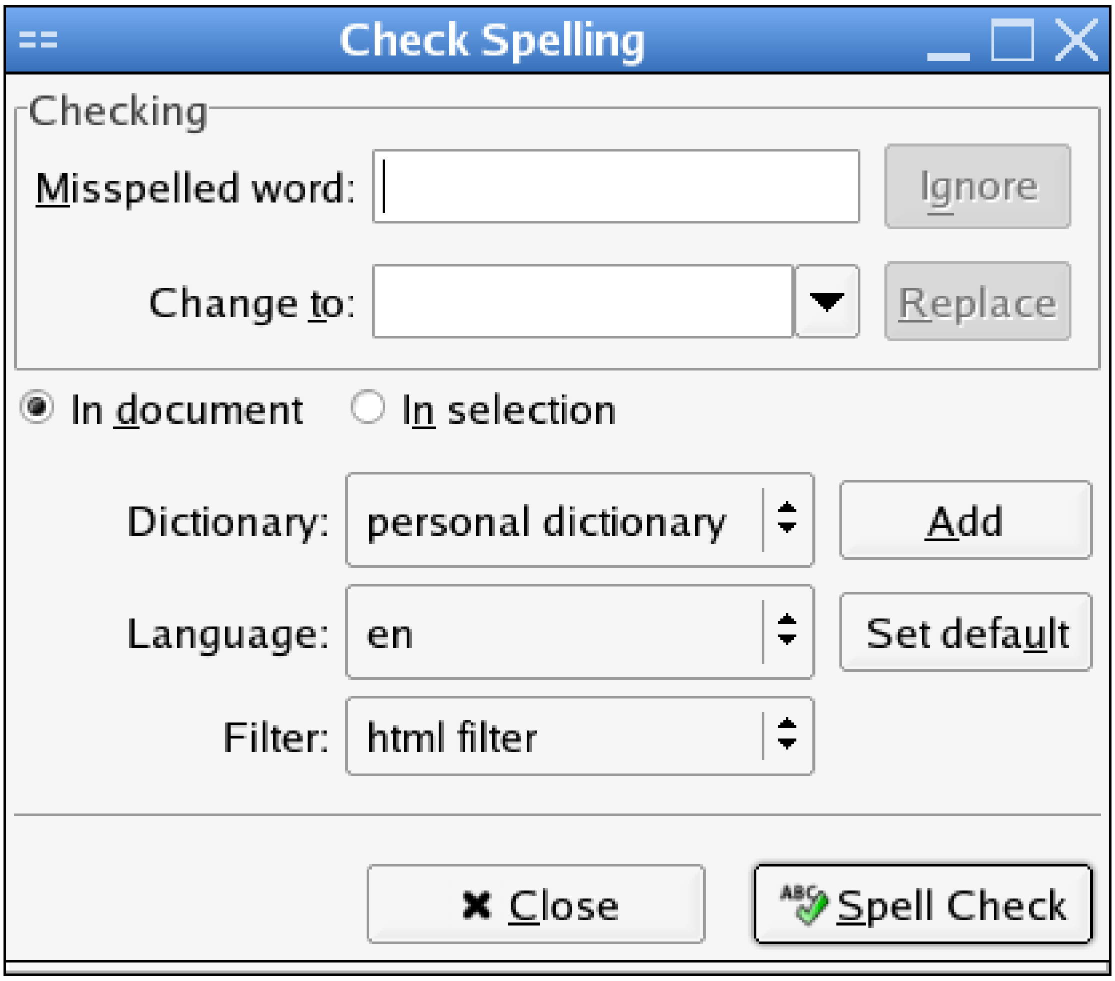


Figure 3.66. Bluefish Spell Checker

Bluefish uses aspell for spell checking. If the aspell libraries are not installed on your system, then the spell checking feature will not be available. At the aspell web site you can also download dictionaries for many different languages.

To launch the spell checker, select Document->Check Spelling... or click on the ABC button on the main tool bar. The spell checker will launch in a separate window, which you can keep open as you edit files.

You have the option to check a whole document or just a selection, to use a personal or a session dictionary, and to choose the language depending on the installed dictionaries.

Click on Spell Check to start spell checking the current document.

You may want to set a default dictionary by first choosing the language in the Language pop up menu, then clicking on Set default.

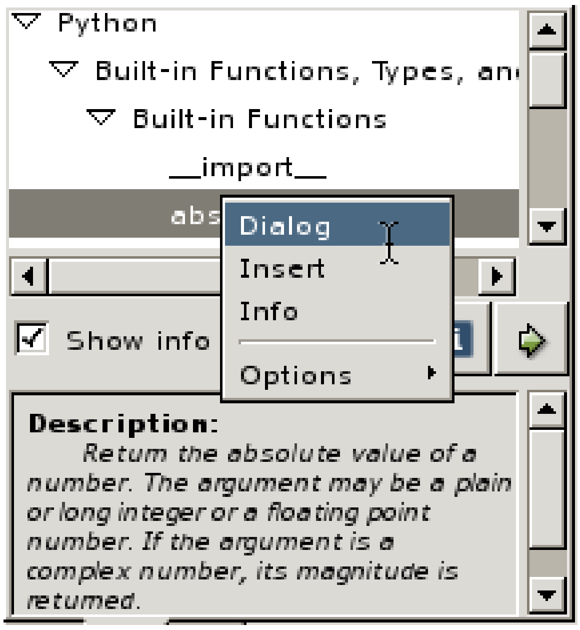
Key words for different languages can be ignored using filters. Currently, the only filter is for HTML. If you want to help write more filters, join the mailing list.

Function reference

The function reference browser contains reference information for different programming and markup languages. Currently, Bluefish comes with a PHP reference, a CSS 2.0 reference, an HTML reference, and a Python reference. The functions are grouped, depending on the language, by type, module, object, etc.

The function reference browser will display an info window on the bottom by checking the Show info window check box. In this window, information about the currently selected item is shown. The type of information shown can be configured in the right-click context menu (see Info Type later in this text).

In the reference browser's contextual menu, you can simply insert the text for the selected item by choosing Insert. Or, you can get a little help using Dialog, which launches a dialog window containing fields for the currently selected item's attributes or parameters. For a summary of an item's usage, choose Info. The contextual menu is also accessible on a group of functions and at the top level of a reference.



Description:
Return the absolute value of a number. The argument may be a plain or long integer or a floating point number. If the argument is a complex number, its magnitude is returned.

Figure 3.67. The reference browser contextual menu

The Options menu accessible via the contextual menu offers three actions:

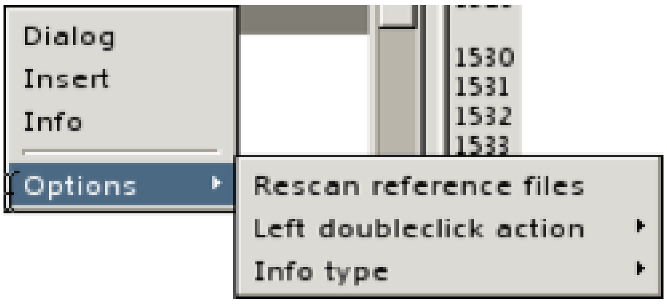
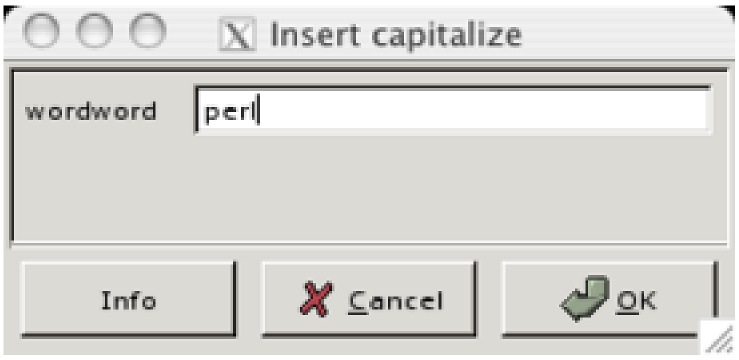


Figure 3.68. The reference browser options menu

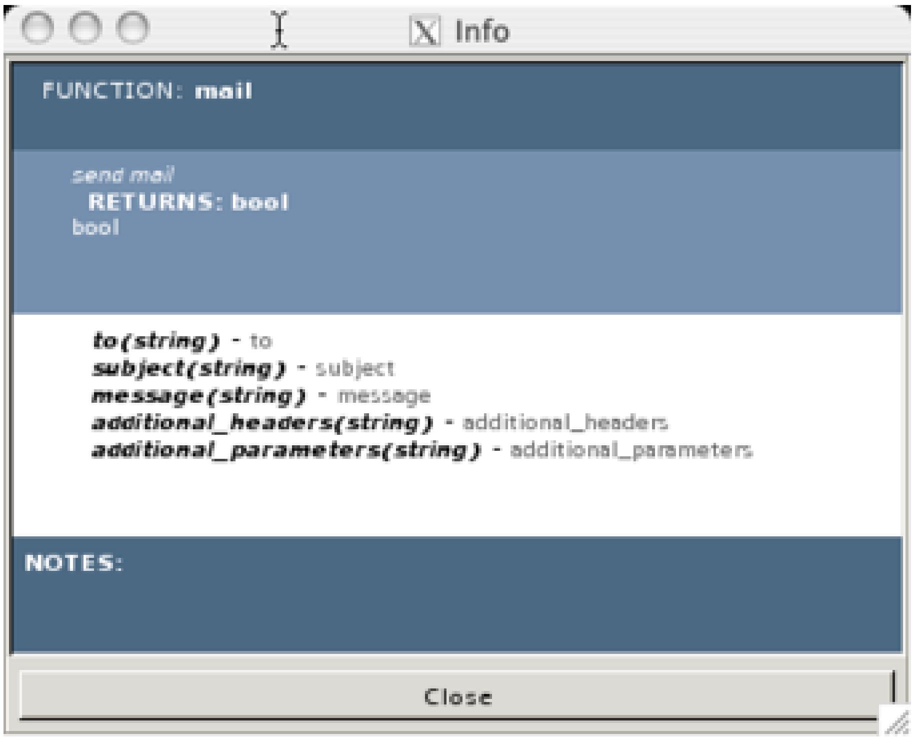
- Rescan reference files in case you have customized one of them, so that the new items be available.
- Left doubleclick action, which can be:
  - Insert to insert the function in the document for latter parametrizing if needed
  - Dialog to insert the function in the document while filling in the parameters in a dialog window:



wordword perl

Figure 3.69. A function reference dialog window

- Info to display a window with all available info about the function:


Info

FUNCTION: mail

send mail
RETURNS: bool
bool

to(string) - to
subject(string) - subject
message(string) - message
additional_headers(string) - additional_headers
additional_parameters(string) - additional_parameters

NOTES:

Figure 3.70. Info available for a function

- Info Type: this is where you can customize what appears in the info window. It can be:
  - the function Description (this is the default)
  - the Attributes/Parameters of the function
  - some Notes about the function

HTML

HTML is obviously the most supported language in Bluefish. There is a special HTML tool bar with many dialogs, and two menus to work with tags:

- the Tags menu:

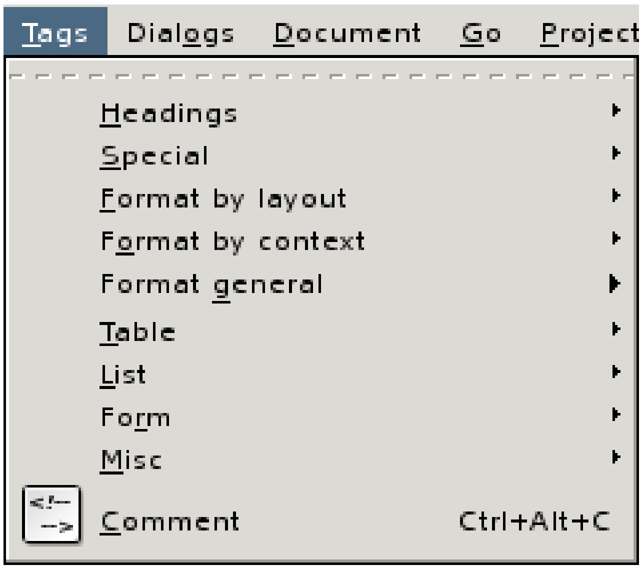


Figure 3.71. The HTML Tags menu

- the Dialogs menu:

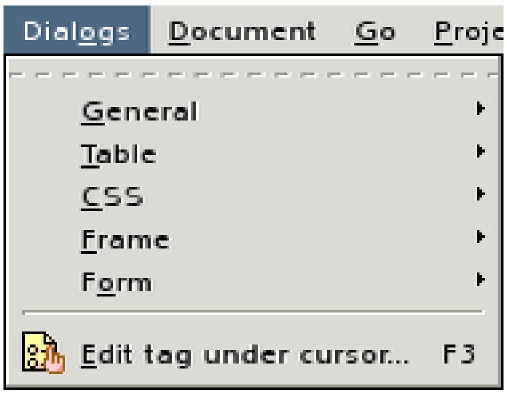


Figure 3.72. The HTML Dialogs menu

The preferences have several settings on HTML style under HTML.

The HTML tool bar has two types of buttons. You can recognise each type by the tool tip if you move the mouse over the button:

- First there are buttons that will open a dialog for some HTML tag. These buttons have a tool tip that ends with three dots.

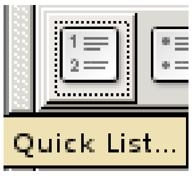


Figure 3.73. An HTML button with a three-dotted tool tip

- Second, there are buttons that will directly insert text, these buttons do not have the dots in the tool tip.

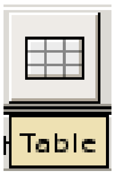


Figure 3.74. A simple HTML tool tip button

If you want to add an HTML tag around some block of text, select the block of text, use the HTML tool bar or the Tags or Dialogs menu to insert the tag. The opening tag will be inserted before the selected block, the closing tag after the selected block.

An existing tag can be edited by right-clicking the tag, and select Edit tag in the context menu. You can also place the cursor in the tag and use Dialogs->Edit tag under cursor... (**F3**). Not all tags, however, have a dialog, so this is not always possible. Colors in the style #RRGGBB can also be edited from the right-click context menu.

In the reference browser on the left panel there is an HTML reference available. All possible attributes and valid values can be found in this reference. See the section called “Function reference” for more info.

Special find an replace features

There are several special search and replace actions in the menu Edit->Replace special. These can be used to convert special characters (like < and &), or ISO characters to their HTML entities, as well as to change the letters case.

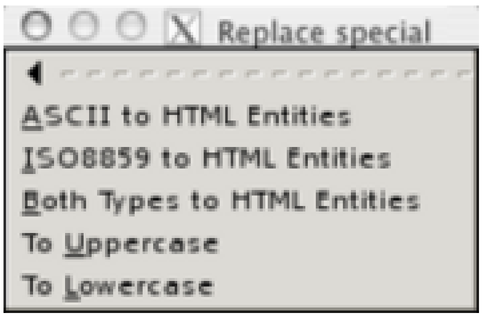


Figure 3.75. The Replace special menu

In all cases, when you want to replace some part of the text, you should first select the part to replace, then use the appropriated menu item.

Thumbnail generation

Bluefish can automatically generate thumbnails for images. A thumbnail is a small image, with a link to the larger image. Bluefish will create the small image based on your settings, and insert a <img> tag in the file, and a <a> tag linking the original. The thumbnails are created in the same directory as the original sources.

The formats used for thumbnails may be png or jpg format. By default, the format used for thumbnails is png. You can change it in the Images panel of Preferences. For jpg images, the thumbnail extension is jpeg.

There are actually two thumbnail dialogs in Bluefish:

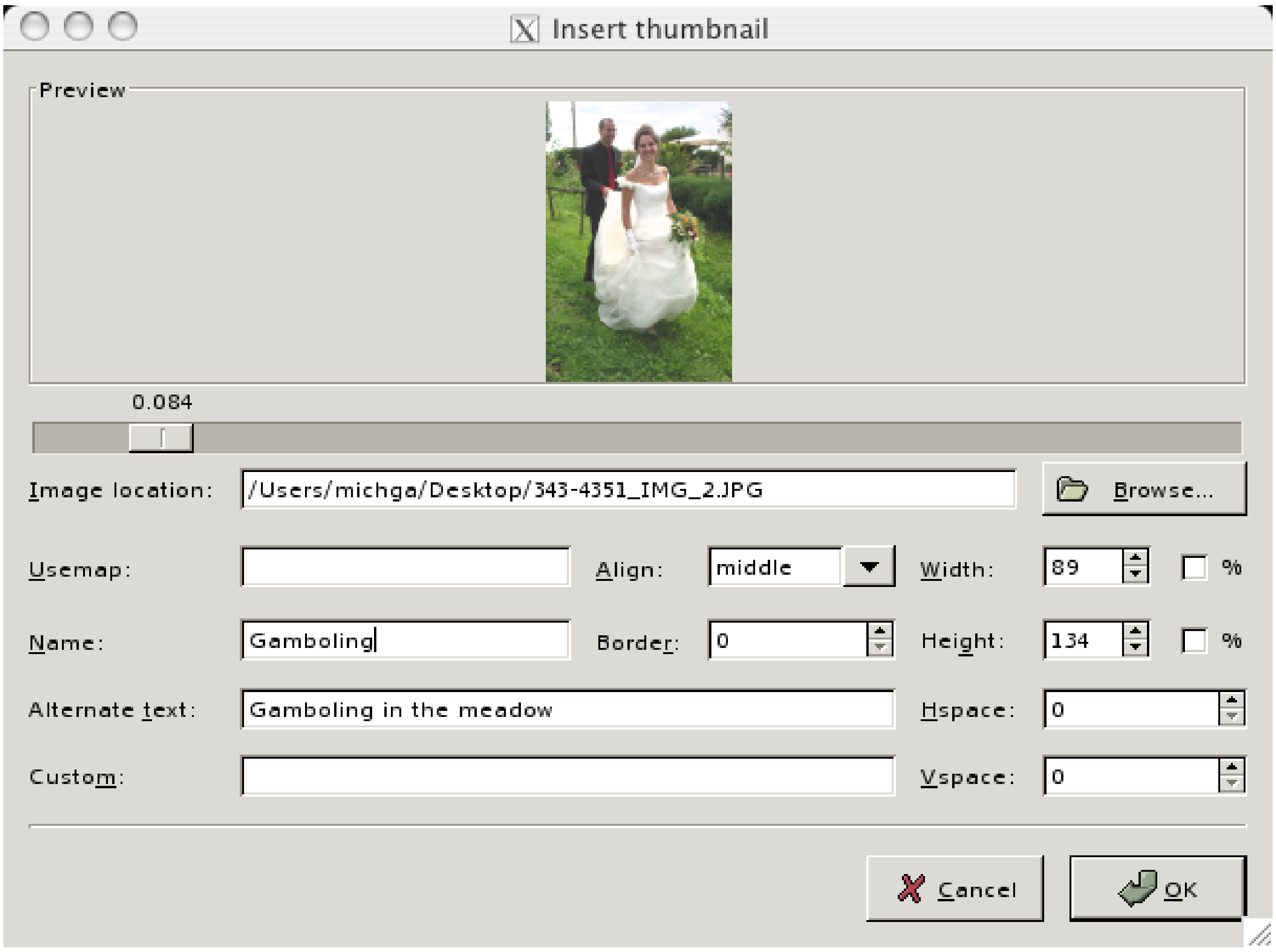
- an Insert thumbnail dialog,accessible from the Dialogs->General->Insert Thumbnail... (**Shift-Alt-N**) or from the Standard bar of the HTML toolbar.

Figure 3.76. The Insert thumbnail icon

- a Multi thumbnail dialog, only accessible from the Standard bar of the HTML toolbar.

Figure 3.77. The Multi thumbnail icon

The Insert thumbnail dialog is very straightforward. You select the image file, provide some <img> tag attributes, choose the scaling, and press OK. The scaling factor is chosen by moving the slider directly under the image. The resulting image is previewed in the preview frame. Bluefish will create the thumbnail with extension _thumbnail.png or _thumbnail.jpeg (depending of the settings for images in the preferences).


Insert thumbnail

Preview

0.084

Image location: /Users/michga/Desktop/343-4351_IMG_2.JPG

Usemap:

Align: middle

Width: 89 %

Name: Gamboling

Border: 0

Height: 134 %

Alternate text: Gamboling in the meadow

Hspace: 0

Custom:

Vspace: 0

Figure 3.78. The Insert thumbnail dialog

Tip

If the source image is not accessible, change webimage to image in the Select File window loaded after clicking on browse for choosing an image. This way you can choose whichever format you want for the original sources. Another way to do it is to change the definition of webimage (see the section called “Modifying file types”).

If that does not solve the problem, it is likely that the type of images you want to load is not defined yet in preferences. In this case, change the definition of image as explained in the section called “Modifying file types”.

As last resource, if you don't want to change the generic file types, you may choose All files in the pop up menu at the bottom of the Select File window.

The code generated for a png image and a png thumbnail looks like this:

```
<a href="/Users/michga/Desktop/343-4351_IMG_2.png">
<img src="/Users/michga/Desktop/343-4351_IMG_2_thumbnail.png"
width="89" height="134" border="0" name="Gamboling"
alt="Gamboling in the meadow" align="middle"></a>
```

and for a jpg image and jpg thumbnail:

```
<a href="/Users/michga/Desktop/343-4351_IMG_2.JPG">
<img src="/Users/michga/Desktop/343-4351_IMG_2_thumbnail.jpeg"
width="89" height="134" border="0" name="Gamboling"
alt="Gamboling in the meadow" align="middle"></a>
```

Note

You can perfectly mix jpg images with png thumbnails or the other way around.

If the html file exists beforehand, the paths to image and thumbnail are inserted relative to the location of the html file. On the contrary, if the html file does not exist beforehand, the full paths to the image and thumbnail are inserted in the code.

In the multi thumbnail dialog, you first choose the scaling method, then you set the corresponding width and/or height parameters. Finally, you may want to adjust the HTML code to be inserted for each image.

Scaling can be based on a fixed ratio, a fixed width, a fixed height, or a fixed width and fixed height (this last option does not keep the original aspect ratio!).

In the HTML code for each image, you can use several placeholders, such as:

- %r for the original filename
- %t for the thumbnail filename
- %w for the original width
- %h for the original height
- %x for the thumbnail width
- %y for the thumbnail height
- %b for the original file size (in bytes)

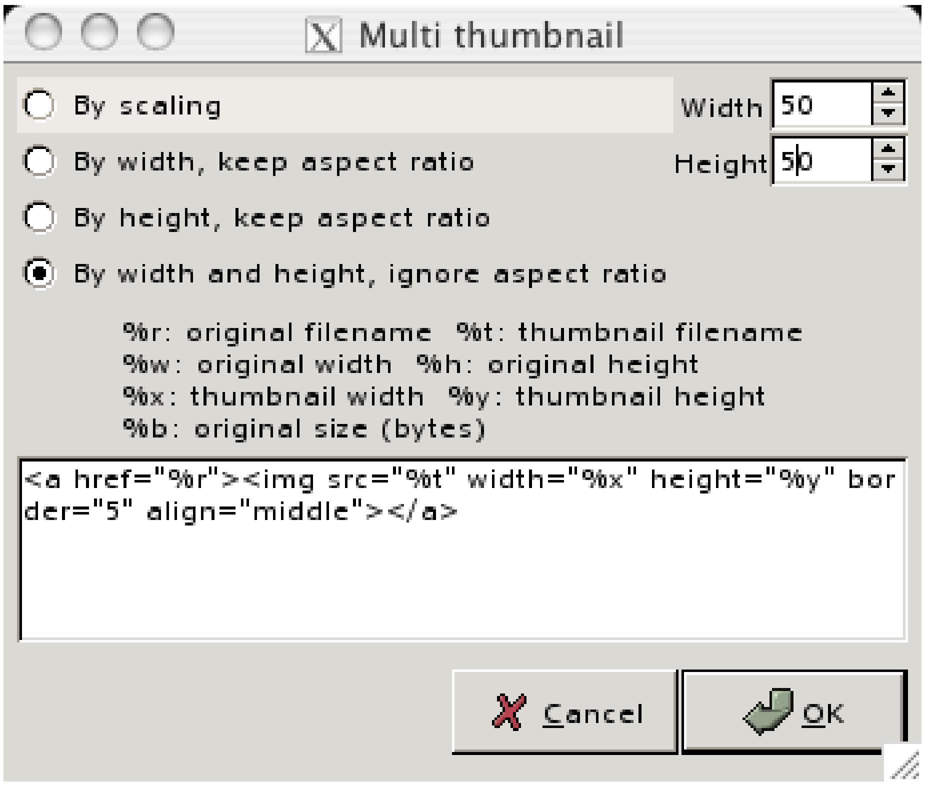
The default string is:

```
<a href="%r"><img src="%t" width="%x" height="%y" border="0"></a>
```

After you have set up the scaling method and parameters, as well as the HTML code, you can select multiple images. Bluefish will create the thumbnails and insert the code.

Here is an example of two thumbnails created with a non nul border width and middle-aligned, with a fixed height and width, disregarding the aspect ratio.

The Multi thumbnail window is the following:



( ) By scaling
( ) By width, keep aspect ratio
( ) By height, keep aspect ratio
(●) By width and height, ignore aspect ratio

Width 50
Height 50

%r: original filename %t: thumbnail filename
%w: original width %h: original height
%x: thumbnail width %y: thumbnail height
%b: original size (bytes)

```
<a href="%r"><img src="%t" width="%x" height="%y" border="5" align="middle"></a>
```

Figure 3.79. The Multi thumbnail dialog

And the generated code is:

```
<a href="/Users/michga/Desktop/tot/343-4351_IMG_2.png">
<img src="tot/343-4351_IMG_2_thumbnail.png" width="50"
height="50" border="5" align="middle"></a>
<a href="/Users/michga/Desktop/tot/343-4352_IMG_2.png">
<img src="tot/343-4352_IMG_2_thumbnail.png" width="50"
height="50" border="5" align="middle"></a>
```

Note

Full pathnames are always used to reference original image sources. The paths to thumbnails are relative to the html file path if the html file already exists, while they are inserted as full paths when the html file does not exist.

Below is a full procedure to quickly generate thumbnails from a directory of image files. This example is purposedly made with deprecated tags, so that you have an idea of what can be made with the variables. Feel free to adjust it when using CSS style sheets.

Procedure 3.4. Generating a photos album with multi thumbnails

1. Put the image files in a folder of their own
2. Open a new file in bluefish, click on the Multi thumbnail... icon in the Standard bar tab of the html tool bar.
3. Enter the scaling percentage in the Scaling (%) field
4. Change the html code as follows:

   ```
   <tr><td><a href="%r">
   <img src="%t"
   width="%x" height="%y" border="0"></a>
   </td>
   </tr>
   <tr><td>Original: %w x %h</td></tr>
   ```

   and click OK.
5. Choose the folder containing the images from the Select files for thumbnail creation window, click **Ctrl**-**A** to select all files, then click OK. The code generated by Bluefish will look like the following:

   ```
   <tr><td><a href="/Users/michga/Desktop/photos/343-4344_IMG.JPG">
   <img src="/Users/michga/Desktop/photos/343-4344_IMG_thumbnail.png"
   width="80" height="53" border="0"></a>
   </td>
   </tr>
   <tr><td>Original: 1600 x 1065</td></tr>
   <tr><td><a href="/Users/michga/Desktop/photos/343-4347_IMG.JPG">
   <img src="/Users/michga/Desktop/photos/343-4347_IMG_thumbnail.png"
   width="80" height="53" border="0"></a>
   </td>
   </tr>
   <tr><td>Original: 1600 x 1065</td></tr>
   ```

6. Use **Ctrl**-**A** to select the file's contents and click on the Table icon located in the Tables tab of the HTML tool bar to embed the code into table tags.

   Figure 3.80. The Table icon in the html tool bar
7. Save the file wherever you want.

If you want to add the file name and the file size in bytes, use this code:

```
<tr><td><a href="%r">
<img src="%t" width="%x" height="%y" border="0"></a>
</td>
</tr>
<tr><td>%r:  %w x %h (%b bytes)</td></tr>
```

Customizing the Quick bar

The Quick bar is a user defined tool bar. All HTML tool bar buttons can be added to the Quick bar by simply right-clicking the button and selecting Add to quickbar.

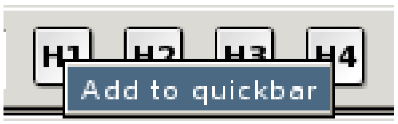


Figure 3.81. Adding an element to the Quick bar

And automagically you will see the element in the Quick bar:

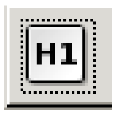


Figure 3.82. The added element in the Quick bar

Note that you cannot add a pop up menu. Thus, if the item you want to add is inside a pop up menu (as is the code tag located in the Context formatting pop up menu of the Fonts tool bar), you have to first click on the pop up menu to display its contents, then to right click on the desired element to insert it in the Quick bar.

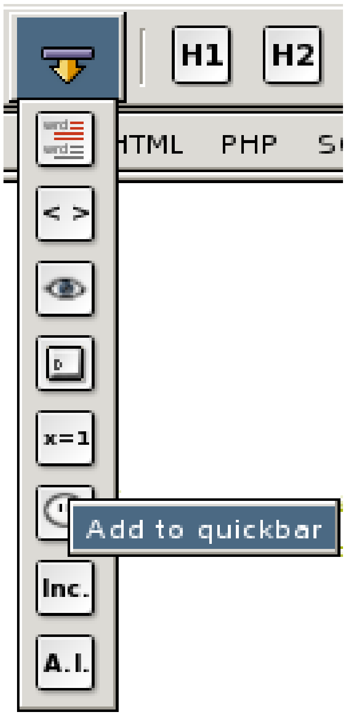


Figure 3.83. Adding a pop up menu element to the Quick bar

If you want to remove items from the Quick bar, right-click them and select Remove from Quick bar.

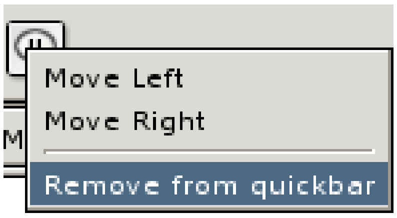


Figure 3.84. Removing an element from the Quick bar

You can also change the location of an element in the Quick bar. To do so, right-click the element and select Shift left or Shift Right as desired. The element will be moved to the left or to the right of its neighborough. Notice that this is not a drag and drop action; you may have to repeat the process if you want to move the element farther.

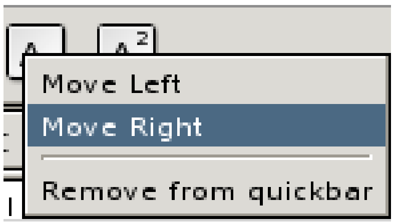


Figure 3.85. Moving an element within the Quick bar

Custom menu

To customize items in the Custom menu tool bar, you will use the Custom menu element:

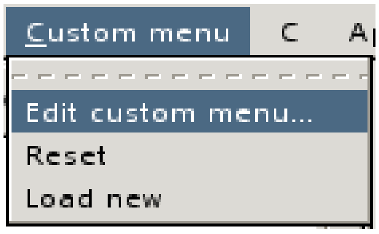


Figure 3.86. Accessing the custom menu

The Custom menu->Edit custom menu... leads to the Custom Menu Editor. The Load new item allows you to load a new menu in case you have directly changed the `custom_menu` file located in the `.bluefish` directory within your HOME directory, while Reset item allows you to return to the default custom menu under the same circumstances.

The `custom_menu` file is created upon install Bluefish and corresponds to some default entries, the ones you can see in the Custom menu tool bar. These will give you an idea what can be done with the custom menu.

The custom menu operates *only* on elements of the Custom menu tool bar, and allows you to:

- add "often used" items to an existing menu
- search and replace patterns to the Replace menu
- create new menus

The Custom Menu Editor is the place where you make all changes to the custom menu. The location for entries in the custom menu is defined by their menu path in the Custom Menu Editor:

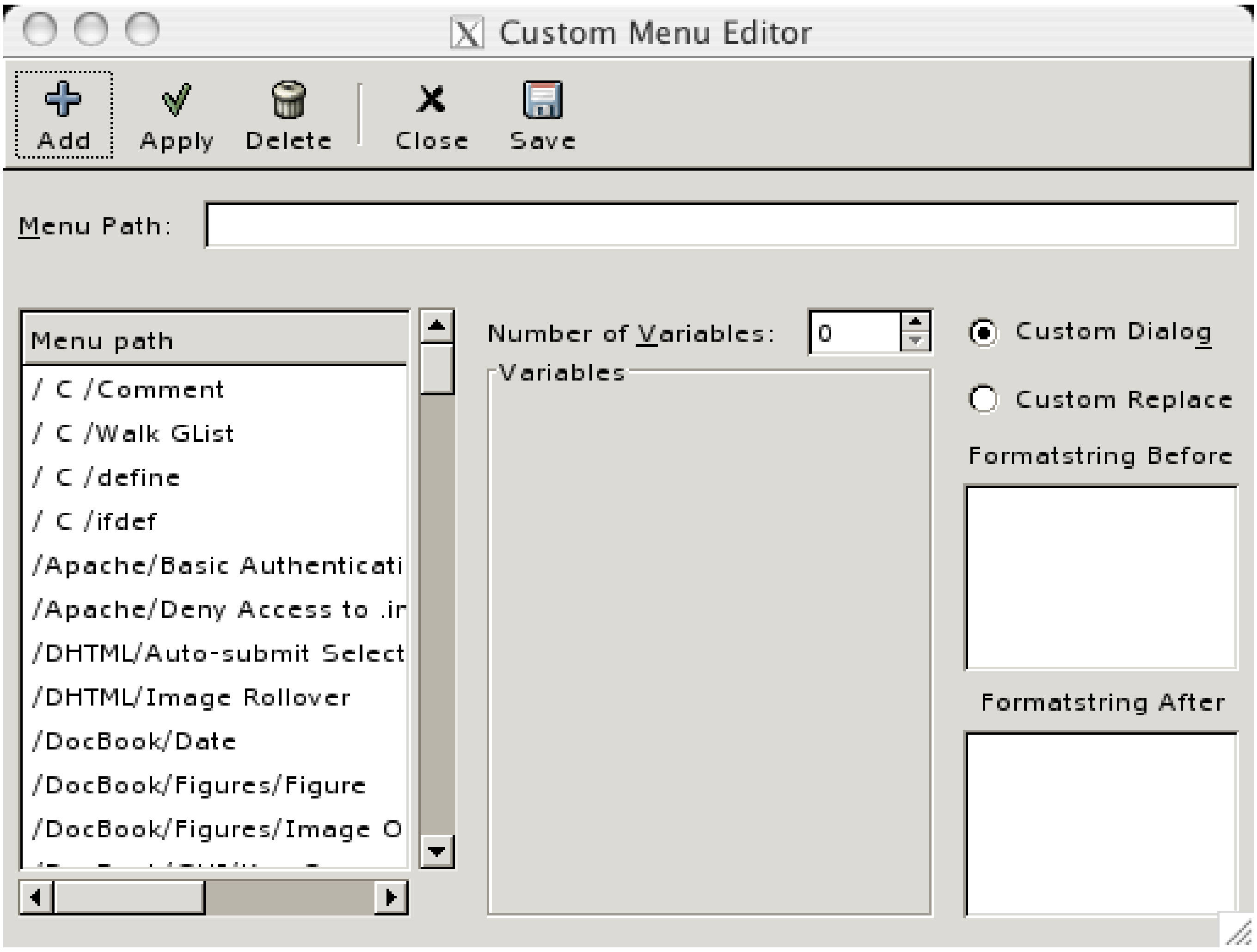
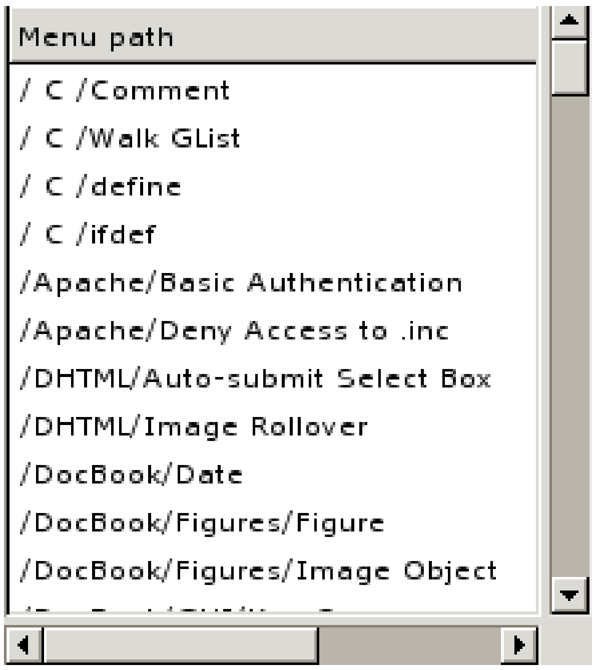


Figure 3.87. The Custom Menu Editor

It has four parts:

- The top one with all action buttons:
  - Add which adds new menu entries, once all necessary fields have been filled in
  - Apply which applies changes to an existing menu entry, once it has been edited
  - Delete which deletes the menu entry currently selected in the Menu path list
  - Close which discards changes
  - Save which saves the changes and exit the editor
- The Menu Path field below the buttons, to enter either an existing or a new menu path
- The Menu path list on the left side, which lists existing menu paths. A menu path looks like */Main menu/submenu/item* or */Main menu/item*. Here's an extract of the default custom menu paths:



Menu path
/ C /Comment
/ C /Walk GList
/ C /define
/ C /ifdef
/Apache/Basic Authentication
/Apache/Deny Access to .inc
/DHTML/Auto-submit Select Box
/DHTML/Image Rollover
/DocBook/Date
/DocBook/Figures/Figure
/DocBook/Figures/Image Object

Figure 3.88. Extract of the default custom menu path

- A custom part on the right side, whose contents changes depending of the type of menu. There are two types of items in the Custom Menu Editor:
  - the Custom dialog, which will insert a string, optionally based on values asked in a dialog
  - the Custom Find and Replace, which will run a replace, also optionally based on values asked in a dialog. Here's how the Custom Replace dialog looks like:

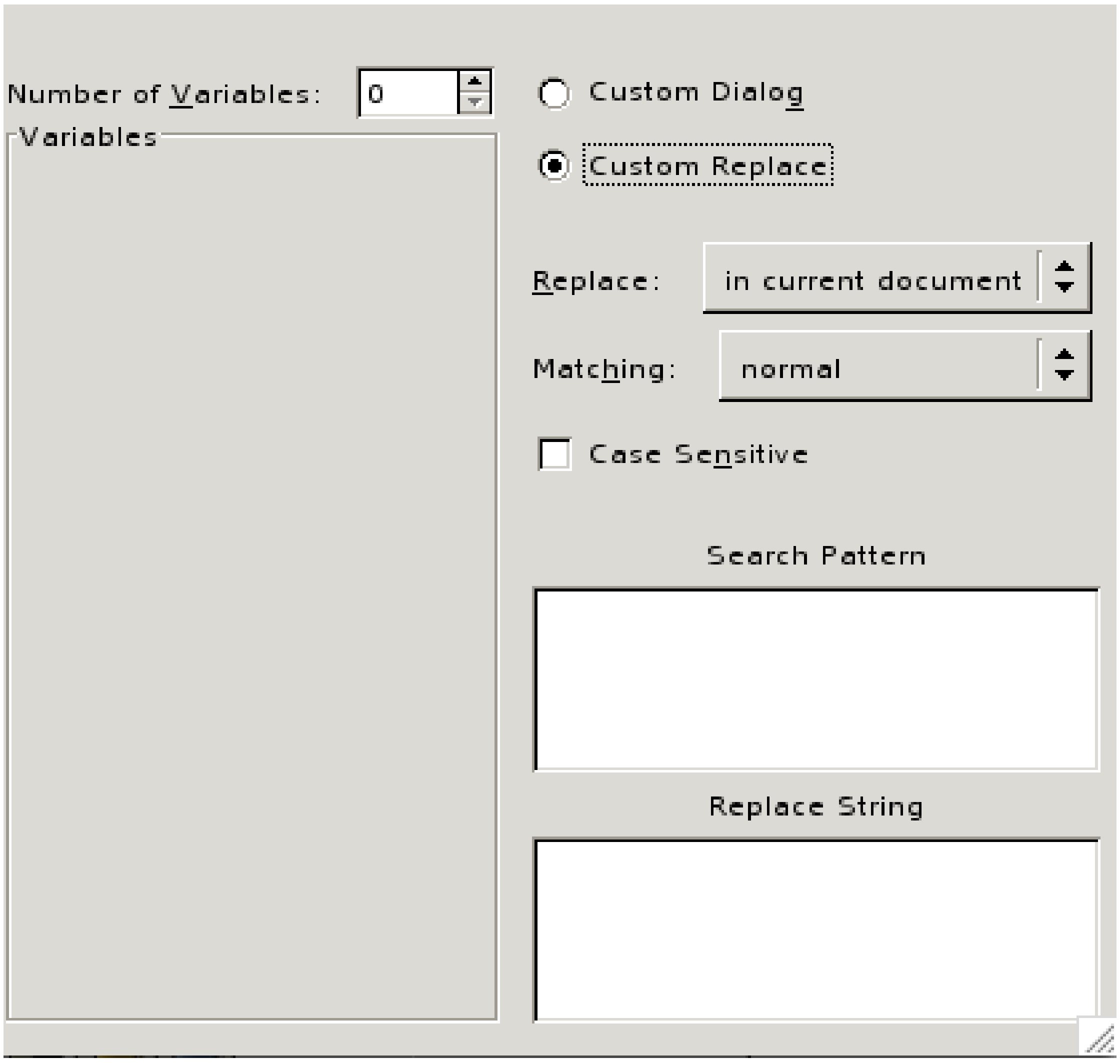


Figure 3.89. The Custom Replace Dialog

The most simple custom dialog item has a menupath, for example */MySite/author*, and a Formatstring before, for example *written by Olivier*. If you add this item, you can add this string by selecting the menu item.

Procedure 3.5. Adding a custom menu based on custom dialog

1. Choose Custom menu->Edit custom menu... in the custom menu tool bar.
2. Enter **`/MySite/author`** in the Menu Path field of the Custom Menu editor.
3. Enter **`written by Olivier`** in the Formatstring Before field located on the right.
4. Click on the Add button at the top.

   Notice that upon adding the new entry, it is listed at the bottom of the Menu path list:

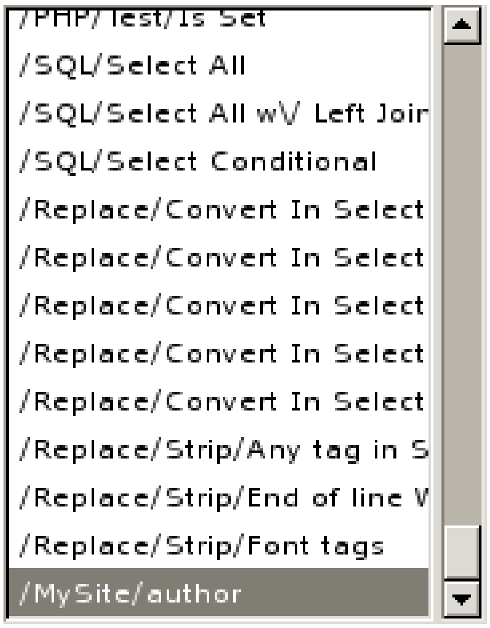

/PHP/Test/Is Set
/SQL/Select All
/SQL/Select All w/ Left Joir
/SQL/Select Conditional
/Replace/Convert In Select
/Replace/Convert In Select
/Replace/Convert In Select
/Replace/Convert In Select
/Replace/Convert In Select
/Replace/Strip/Any tag in S
/Replace/Strip/End of line V
/Replace/Strip/Font tags
/MySite/author

Figure 3.90. A new custom entry in the Menu path list

5. Click on the Save button. This will add the menu to the Custom menu tool bar:

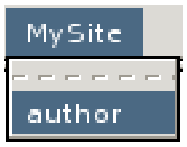


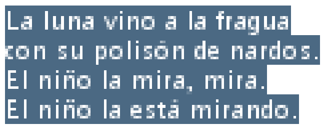
Figure 3.91. A new menu in the custom menu tool bar

Note that the new menu is placed at the right end of the custom menu tool bar. When closing Bluefish and relaunching it, it will be placed in alphabetical order, except that the Replace menu will always be at the far right side.

In another example, you have a string you often need to set before and after some block of text, for example <div class="MyClass">YourBlockOfText</div>. To do it:

1. Open the Custom Menu Editor
2. Enter **`/MySite/div with class`** in the Menu Path field
3. Enter <div class="MyClass"> in the Formatstring Before field
4. Enter </div> in the Formatstring After field
5. Click on Add, then on Save. The item will appear in the menu.

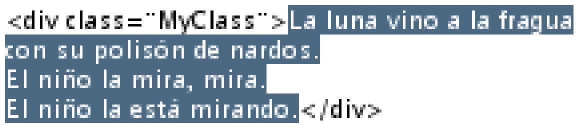
If you now select some text:



```
La luna vino a la fragua
con su polisón de nardos.
El niño la mira, mira.
El niño la está mirando.
```

Figure 3.92. A block of selected text before activating the menu

And activate this menu item, you will see that the first bit of text is added before the selection, and the second bit is added after the selection:



```
<div class="MyClass">La luna vino a la fragua
con su polisón de nardos.
El niño la mira, mira.
El niño la está mirando.</div>
```

Figure 3.93. A block of text after activating the menu

Suppose you want to improve this last example. You have both MyClass1 and MyClass2 and want to be able to choose the desired class when activating the menu. Here's how to do it:

1. Open the Custom Menu Editor
2. Browse the Menu path list to retrieve the /MySite/class with div entry and click on it to make appear its components in the Menu Path and Custom Dialog fields
3. Click on the top arrow of the Number of Variables popup menu to get 1 in the field. As you see a Variables entry appears where you can enter the name for variable %0. As name we enter *MyClass number*
4. Now change the FormatString Before field to take this new variable into account, as following: <div class="MyClass%0">
5. Click on Apply so that your changes will be taken in account, and click on Save to update the menu.

If you now activate this menu after having selected a block of text, you will be presented with a new dialog asking you for the value of *MyClass number*:

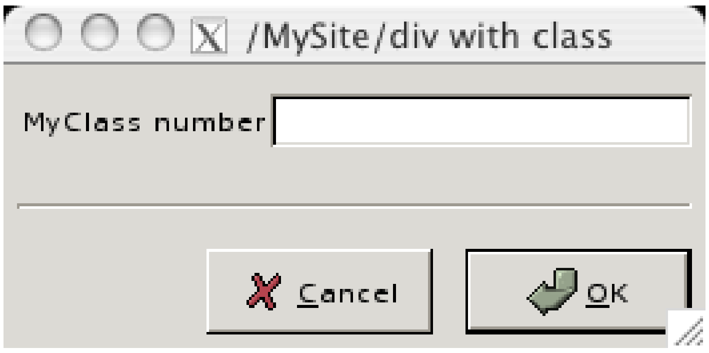
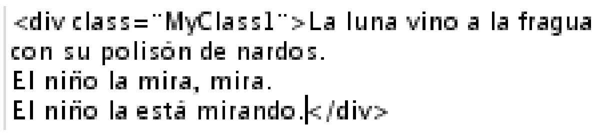


Figure 3.94. The new div with class dialog

After entering the desired value, the same process as before will occur, using the value you provided. Here we have entered 1 as value:



```
<div class="MyClass1">La luna vino a la fragua
con su polisón de nardos.
El niño la mira, mira.
El niño la está mirando.</div>
```

Figure 3.95. The block of text after entering the value

Tip

you can use the **Return** and **Tab** keys to format the output.

Any variable can be used any times you want in the dialog.

Find and replace items are no different. The dialog has some more options, each of these options correspond to the regular replace dialog. Again you can use variables like %0, %1 etc. to make a certain menu item more flexible.

SCREENSHOT BY ????

External programs, filters

You can integrate external commands such as browsers, or text filters. If you want to use for example a sed command as a filter, you can add it like this to the external commands and filters (in the preferences dialog): sed -e 'some sed command' > '%f' < '%s'

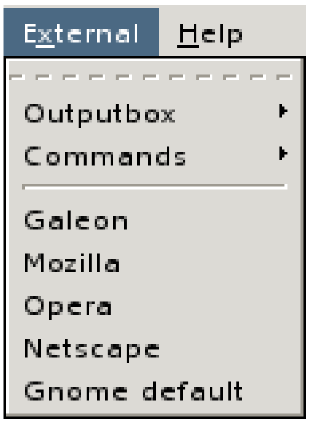


Figure 3.96. Bluefish External Menu

To add items to the external menu:

1. Edit > Preferences
2. Choose the "External Programs" tab.
3. Click "Add" to add a new item
4. Double-click on the name field to give the command a name (this name will become an option in the "External" menu).
5. Double-click on the command field to type the command to be executed.

I WILL ADD AN EXAMPLE HERE LATER

Items within External > Outputbox allow for programs to give feedback by opening an output box within Bluefish's main window. To create an Outputbox menu item:

1. Edit > Preferences
2. Choose the "Output parsers" tab.
3. Click "Add" to add a new item
4. Double-click on the name field to give the command a name (this name will become an option in the "External" menu).
5. Double-click on the pattern field and give a regular expression pattern with subpatterns, such as line ([0-9]+) column [0-9]+ - (.*).
6. Double-click on the "File #" field and give the number for the subpattern matching the filename (-1 for none)
7. Double-click on the "Line #" field and give the number for the subpattern matching the line number
8. Double-click on the "Output #" field and give the number for the subpattern matching the actual error message
9. Double-click on the "Command" field and the command to execute, %s is the current filename
10. Toggle the "Show all output" check box to show output NOT matching the regular expression.

Note: Of course, it is also possible to add these items by editing the file named "rcfile_v2" found in the user's home directory (~/.bluefish/rcfile_v2). The fields are delimited by colons and correspond to those found in the GUI.

Customising Bluefish

Most of customization can be made through the Edit preferences panel, which is divided into several tabs.

Modifying shortcut keys

Many menu entries are accessible via key combination; also called a shortcut. For example, pressing the **Ctrl**-**S** keys saves the current file to disk. If available, shortcut key combinations are shown on the right of the menu entry.

What many people do not know is that they can be changed. Move the mouse over a menu entry, and press the key combination you would like to use. Immediately this combination will show up on the right of the menu entry. An entry can also be removed, press the backspace key when you move the mouse over a menu entry to remove the shortcut.

To save the shortcut key combinations for later Bluefish sessions, use Edit->Save Shortcut Keys. This will store the settings in the `~/.bluefish/menudump_2`. If you want to restore the default combinations simply remove this file and restart Bluefish.

Editor preferences

The font of the editor can be set in the preferences under Editor.

A frequently asked question is how to change the background color of the editor in Bluefish. Bluefish uses the default editor background color as set in your GTK theme. If you want to override the color of the theme, edit ~/.gtkrc-2.0 and add this section:

```
style "EditorStyle" {
        base[NORMAL]="#eeeeee"
}
class "GtkTextView" style "EditorStyle"
```

Obviously, #eeeeee should be your preferred background color.

Modifying file types

Here you can define all file types that should be recognised by Bluefish. The defaults for these file types are retrieved from a file called file types.default in ${prefix}/share/bluefish/.

The file types consist first of a name (this name is also used in the file filters, and in the highlighting patterns). Second is a list of extensions, separated by a colon (:). Third are the highlighting update characters. Upon a key press of one of these characters, the highlighting engine will refresh the highlighting around the cursor. If this field is empty, any character will force the highlighting engine to refresh. Special characters like the tab and the newline can be entered as \t and \n, the backslash itself is entered as \\. Fifth is the icon location for this file type. Sixth is whether this file type is editable by Bluefish (whether or not Bluefish should try to open it after a double click). Seventh is a regular expression that can be used to detect the file type if a file without extension is loaded. Eight is the auto-tag-closing mode. A value of 0 means that Bluefish should not close XML/HTML tags, a value of 1 means it should close the tags XML style (<br />), a value of 2 means HTML style.

Modifying the files filters

TO BE WRITTEN

Modifying the highlighting patterns

The highlight patterns are build from Perl compatible regular expressions. A pattern has options for coloring and style of the text it matches. Within a match other patterns can be used to color parts of that match. There are three types of patterns:

- 1 Two patterns, match from the start to the end pattern
- 2 One pattern that matches from start to end
- 3 Match a subpattern from the parent pattern

One specific pattern can also be used within several other parent patterns. The parent-match option is a regular expression that defines all parents for a certain pattern. If empty it will default to ^top$, so basically it will be on the top level.

So how does it work? Lets take a look at a little example text, a piece of PHP code within some HTML code:

```
<p align="center">
<?php
// this is a comment ?>
?>
```

The first thing the highlighting engine does is finding the pattern that has the lowest match. Using the default patterns for PHP, the pattern named `html` has a match at position 0:

```
<p align="center">
```

So now the highlighting engine searches for the lowest match in all subpatterns of `html`, in the region matched by the type 2 `html` pattern. Again, the lowest match will count. The pattern named `html-tag` has a match at position 1. This pattern is a type 3 pattern, so it matches a subpattern of the parent:

```
p
```

the match from subpattern `html-tag` ends at position 2 and it does not have any child patterns, so the highlighting engine continues at position 2 with all subpatterns from `html`. A type 2 pattern named `html-attrib` has the lowest match:

```
align="center"
```

This pattern does have a child pattern, again a type 3 pattern called `html-attrib-sub2` matching:

```
"center"
```

The pattern `html-attrib-sub2` does not have any child patterns, and subpatterns of `html-attrib` do not have any more matches, and also `html` subpatterns do not have any more matches. So we are back on the main level, the remaining code to highlight is:

```
<?php
// this is a comment ?>
?>
```

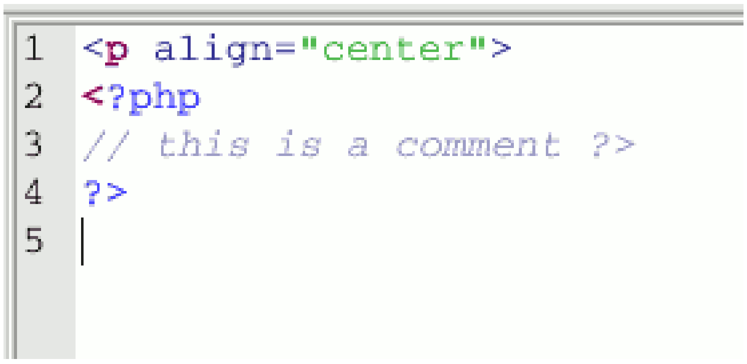
Now a pattern named `php` has the lowest match. This is a type 0 pattern, so the highlight engine continues with all the remaining code, but it will not only search for the lowest match of the child patterns of `php`, but it will also use for the end pattern of `php`. The lowest match in this example is a pattern named `php-comment-C++` As you can see the ?> within the comment does not end the `php` pattern, because it lies within a subpattern of `php`:

```
// this is a comment ?>
```

The pattern `php-comment-C++` does not have any child patterns, so the remaining code for the `php` subpatterns is:

```
?>
```

It is very obvious now, the lowest match will be the end pattern of the `php` pattern, so we're back on the main level, and we have matched all of the code!



```
<p align="center">
<?php
// this is a comment ?>
?>
```

Figure 3.97. Syntax highlighting example

The config file for highlighting is a colon separated array with the following content:

```
mode:
patternname:
case_sensitive(0-on/1-off):
start reg-ex:
end reg-ex:
start & end pattern(1), only start(2), subpattern(3):
parent-match:
foreground-color:
background-color:
don't change weight(0), non-bold(1), bold(2):
don't change style(0), non-italic(1), italic(2):
```

The same options are found in the syntax highlighting preferences.

SCREENSHOT BY ????

Chapter 4. Debugging Bluefish

Using the Debugger

Here are the detailed steps for sending a useful backtrace to the Bluefish Developer Team.

Procedure 4.1. Running bluefish under gdb

1. Get the latest CVS release (see the section called “Latest Developmental Version” for info)
2. From `bluefish-gtk2`, the top directory of the bluefish source, run: **`autoconf`**

   You may have to set some environment variables before running **autoconf**, as well as providing autoconf with some flags. Also, some patches may need to be applied, depending on your system.
3. Then, run **`./configure --with-debugging-output`**

   Again, you may have to add some flags, depending on your system.
4. Once you succeed in configuring bluefish, run: **`make clean`** in order to remove all unnecessary files.
5. Then, run **`make`** to compile bluefish.

   Do not run make install since it strips the debugging symbols from the executable.
6. Execute bluefish under gdb with: **`gdb src/bluefish`**. This way, you will get access to a non stripped version of bluefish, which is not the case if you run **`gdb bluefish`** or **gdb /usr/local/bin/bluefish**, since those binaries do not have any debugging symbols anymore.
7. Once **gdb** has started, type **`r`** to start the debugging session.
8. Try to reproduce the crash in bluefish.
9. Copy and paste the last 50 lines of debugging output to an email.
10. Type **`bt`** in gdb to get the backtrace, and copy it to the email too. If the backtrace is huge, copy only the first 50 lines.
11. Send the email to the general address, the mailing list, or a specific developer (see the section called “Contact Us” for info).
12. Quit gdb with **`q`**

Chapter 5. Reference

... list all options in the preferences and their config file and config-name

Chapter 6. Development guidelines

Work hard but have fun!

Indenting and formating style

Indenting can be done with the **indent** command line tool. Bluefish uses tabs - not spaces, and I'll explain why.

Some programmers prefer a lot of indenting, 8 characters, some prefer less, 3 characters. If Bluefish code was indented with spaces, these programmers had a problem, they would have to change the files to view it in their favourite layout. But because we use tabs, these programmers can simply set the tab width to a different value, and without changing the files it looks good for both programmers!

To indent properly with **indent**, issue this command:

```
$ indent --line-length 100 --k-and-r-style --tab-size 4 \
-bbo --ignore-newlines bluefishcode.c
```

Comment all public functions like it is done in `bf_lib.c` and `gtk_easy.c` (javadoc style, with some small differences), this can be used to create a function reference.

Naming

For non-local functions, the name should preferably include a prefix that shows the part of bluefish it is used for. There are, furthermore, many often used abbreviations in the bluefish code, such as:

Abbreviations used in the Bluefish code

doc	A function for handling a specific document
bfwin	A function for handling a specific Bluefish window
cb	Callback, a function called after a button click or some other event
lcb	Local callback, a function called after an event, only used in this .c file

Here are such function names that show where they are from, what they handle, and/or what they do

Examples of function names in Bluefish

bmark_set_for_doc	Bookmark code, sets bookmarks for a document
spell_check_cb	Spell check code, this is a callback function (for a button)
project_open_from_file	Project code, opens a new project from a given file name

Declaring procedures

All local functions should be static!

Callback functions (called for events such as button clicks) should have prefix *_cb*, or *_lcb* for local callbacks.

For GTK callback functions, use the name of the signal in the name.

Header files

Only functions that are used from outside the file itself should be in the header file, in the order in which they are found in the .c file itself. Basically these are all non-static functions in the .c file.

New files

About new files

File reference

References

Patches

Before starting to code:

1. Update your CVS tree, or alternatively download the latest snapshot
2. Copy this original tree, so you can make a patch against this tree

Before creating the patch:

1. Run **`make distclean && ./configure && make`** and test if it runs successfully
2. If you have the possibility do this both with `gcc-2.95` and `gcc-3.x` as compiler

Now create the patch. Assuming that you have two directories, `original-tree` and `my-tree`:

1. Run **`make distclean`** in both trees
2. cd to the parent directory of both trees
3. Run **`diff -Naur original-tree my-tree | bzip2 -9c > patchbla.diff.bz2`**

Translations

Introduction

Bluefish has been translated into more than 15 different languages and this is only the beginning.

Translation process is not a difficult task but you will need some time because there are more than one thousand strings to be translated. The good news are you don't need to be a programmer to make Bluefish speak your language and the only tool you need is a text editor (Vim, Emacs, bluefish, etc.)

Bluefish uses po (Portable Object) files. A po file is just a plain text file that you can edit with your favorite text editor.

PO files basics

In a typical po file there are five major types of entries:

1. Those which begin with "#:" showing the places in the source code that contains the string being translated (there may be one or more) as: '`#: ../src/about.c:123`'
2. Those which begin with "#," containing some flags (not always present) as: '`#, c-format`'
3. Those which begin with "msgid" containing the English string being translated (it may be spanned in several lines) as: '`msgid "Authentication is required for %s."`'
4. Those which begin with "msgstr" containing the translated string as: '`msgstr "Une autorisation est requise pour accéder à %s."`'
5. Those which begin with "#~ " containing obsolete strings as: '`#~ msgid "Save document as"`'

Warning

When an entry is tagged as fuzzy (i.e. when the line begins with "`#, fuzzy`"), that means it is probably incorrect.

You have to make sure the translation is correct and then delete either the "`#, fuzzy`" line if this is the only flag on the line, or "`, fuzzy`" part of the line if there are some other flags on the same line, like in "`#, fuzzy, c-format`".

Remember that as long as a translation is marked "fuzzy", it will NOT actually be used!

As far as obsolete strings are concerned, it is up to you to decide if you want to remove them. On one hand they can be reused in a latter version of the po file, on the other hand they make the po file bigger.

Hence, your task as a translator is to:

1. Translate all empty msgstr entries
2. Check all fuzzy entries, correct them if they are wrong and remove all fuzzy tags
3. Optionally, remove obsolete strings

4. Check that the po file ends with a blank line

Shortcut keys

Shortcut keys, known as hotkeys or even accelerator keys, are defined as follows (look at the underscore, please):

```
# src/toolbars.c:482
#: ../src/filebrowser.c:1453
msgid "/_Refresh"
msgstr "/_Actualizar"
```

That it means that in the English locale the user have to press **Alt-R** to activate this particular GUI element. On the other hand if your locale is Spanish your shortcut key will be **Alt-A**.

Warning

You have to keep in mind that two GUI elements must not have the same shortcut key at the same level.

How to contribute

It is really easy. Just drop me a line at `<wecharri(at)arnet.com.ar>` and I will send you your po file ready to be translated. When you have finished the translation work, just send it me back (use gzip or bzip2 if possible, please). Then I will check it and if everything is right I will add it into CVS.

About ten days before a new release I will send a new fresh po copy to each translator to repeat this process.

All po files will be named as follows:

```
date-foo.po.gz (date: day-month-year)
```

Example:

```
12-12-2004.es.po.gz (for Spanish po file)
```

Please, remember:

- Do not rename it (I need it for tracking stuff)
- Send it me back as soon as possible in zipped format too.
- Do not mix it without any local copy you have.
- Remember they are in UTF-8 format
- Subject in my mail will be ***New Bluefish PO File !***

And at last, do not start a new translation before contacting me or contact Olivier and do not post your po file at the list, please.

If you have some doubts, do not hesitate contact me at `<wecharri(at)arnet.com.ar>`.

Some tips

Development tips

Making releases

What is to do

Useful stuff

Links and so

Appendix A. Credits

Bluefish developers

Here are the developers for release 1.0:

- Olivier Sessink
- Jim Hayward
- Oskar Swida
- Eugene Morenko
- Alastair Porter

Developers for previous releases are:

- Chris Mazuc
- Neil Millar
- Gero Takke
- Bo Forslund
- David Arno
- Pablo De Napoli
- Santiago Capel Torres
- Rasmus Toftdahl Olesen
- Roland Steinbach
- Christian Tellefsen
- Antti-Juhani Kaijanaho

Bluefish package maintainers

The following people maintain bluefish packages for various systems:

- Debian: Davide Puricelli
- Redhat: Matthias Haase
- Mandrake: Todd Lyons
- Fink: Michèle Garoche

Bluefish translators

Translators for the 1.0 release are:

- Brazilian Portuguese - Anderson Rocha
- Bulgarian - Peio Popov
- Chinese - Ting Yang (Dormouse)
- Danish - Rasmus Toftdahl Olesen
- Finnish - Juho Roukala
- French - Michèle Garoche
- German - Roland Steinbach
- Hungarian - Péter Sáska
- Italian - Stefano Canepa
- Norwegian - Christian Tellefsen
- Polish - Oskar Swida
- Portuguese - Lopo Pizarro
- Russian - Eugene Rupakov
- Serbian - Marko Milenovic
- Spanish - Walter Oscar Echarri
- Swedish - David Smeringe
- Tamil - Murugapandian Barathee

Supporters to bluefish

Supporters

Appendix B. Bluefish change history

History

Changes in release GTK2-port

To be written

Changes in release GTK1-version

To be written

Appendix C. Guidelines for Writing this Manual

Introduction to DocBook

The Bluefish manual is written in DocBook XML, which is a set of standards for writing documentation. Originally, DocBook was intended for computer software documentation, but is now used for many other document types.

Building the Manual

To generate HTML, PDF or PostScript files out of the source XML, you will need the following:

- the Bluefish manual source files via CVS
- the Bluefish doc tools via CVS
- DocBook 4.2.0
- DocBook XSL style sheets
- XSLT Processors: we use **xsltproc** for HTML production, and **fop** for PDF and PostScript production.
- **xmllint** for validating all files

Here are the procedures to install the required files:

Procedure C.1. Getting the Bluefish manual source files

1. Login:

   ```
   $ cvs -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/bluefish \
           login
   ```

2. Hit enter at the prompt when asked for your password.
3. Checkout the directory containing the Bluefish documentation:

   ```
   $ cvs -z3 -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/bluefish \
           co bluefish-gtk2/doc
   ```

   This will download all the files to your system in the directory `bluefish-gtk2/doc/`.

Procedure C.2. Getting the Bluefish doc tools

- Get the doc tools from CVS under `bluefish-doctools`:

  ```
  $ cvs -z3 -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/bluefish \
          co bluefish-doctools/
  ```

Procedure C.3. Installing docbook-xsl

1. Install them for your distribution
2. Put a copy of the unpacked tarball into `bluefish-doctools/tools`, renaming it xsl.

Procedure C.4. Installing the xslt processors

1. Install libxslt if needed.

   xsltproc is provided by the libxslt, distributed as part of the GNOME desktop environment and is packaged for most Linux distributions. Fink provides the package for Mac OS X.
2. Optionally, install fop for your distribution.

Conventions for Writing this Manual

We recommend to use bluefish to write the manual. It has most of the tags used in the manual in the DocBook custom menu. By unchecking the Use spaces to indent, not tabs and checking Word wrap default in the Preferences Editor panel, you will ensure that no unnecessary white space will be produced when processing the files.

The DocBook rules are strict and must be maintained in order for the manual to build using `xsltproc`. Thus, you should always validate the whole book before sending or committing any change. To do it, just issue:

```
$ make validate-all
```

in the `bluefish-gtk2/doc/source` directory.

However, there are some rules we like to follow to make editing the manual more efficient and organized.

The `id` Attribute

We use `id` on chapter, appendix, section, figure, and procedure. This provides a convenient way to reference them in the text as well and to get them listed in the table of contents.

If you need to reference some chunk of text embedded in a tag different from those already mentionned, you can also use an id on this tag, since DocBook allows id on all tags.

All `id` names should begin with the name of the book. In our case the main book file is *bluefish.xml*, so every `id` should begin with *bluefish*.

Separate words in the `id` with hyphens.

Finally, include a word or two describing the content in the section. For example, a chapter entitled `Using Bluefish` would have the id *bluefish-using*. And, a section within that chapter called `Keyboard Shortcuts` could have the name *bluefish-using-shortcuts*.

The main thing is that all id's must be unique or processing will fail. To ensure that all id's are unique, just run **make validate-all** before committing the changes.

Also, be careful when renaming id's, since the name could be used in links within other parts of the manual. It's best to do a global search for an id in all the manual's files before changing an id.

Using Screen shots

All screen shots are png files. They should be placed in the `bluefish-gtk2/doc/source/figures` directory. They are inserted in the xml files with the following tags:

```
<para>
<figure id="figure-file-menu">
<title id="figure-file-menu-title">Bluefish File Menu</title>
<screenshot>
<mediaobject>
<imageobject>
<imagedata fileref="figures/file_menu.png" format="PNG"/>
</imageobject>
<textobject>
<phrase>A screen shot of the Bluefish File Menu</phrase>
</textobject>
</mediaobject>
</screenshot>
</figure>
</para>
```

Notice that the `figure id`, the `title id`, and the `imagedata fileref` are very similar. The former ones use hyphens, while the later uses underscore to separate the `id` parts. They have in common the significant part. Do not forget to put in the `phrase` tag, a sentence meaningful for blind people.

Referencing Bluefish interface elements

We use the following DocBook GUI tags:

Interface elements

Isolated shortcut

```
   <keycombo>
   <keycap>Ctrl</keycap>
   <keycap>S</keycap>
   </keycombo>
```

Isolated menu

```
<guimenu>File</guimenu>
```

Menu with submenuitem

```
<menuchoice>
<guimenu>File</guimenu>
<guimenuitem>Open...</guimenuitem>
</menuchoice>
```

Menu with submenu

```
<menuchoice>
<guimenu>Edit</guimenu>
<guisubmenu>Replace special</guisubmenu>
</menuchoice>
```

Menu with submenuitem and shortcut

```
<menuchoice>
<shortcut>
<keycombo>
<keycap>Ctrl</keycap>
<keycap>O</keycap>
</keycombo></shortcut>
<guimenu>File</guimenu>
<guimenuitem>Open...</guimenuitem>
</menuchoice>
```

Label

```
<guilabel>Use spaces to indent, not tabs</guilabel>
```

Using procedures

When you want to explain some process, use procedures; this way, the user benefits of a clear step by step guidance:

```
<procedure id="installing-docbook-xsl">
<title>Installing docbook-xsl</title>
<step>
<para>Install them for your distribution</para>
</step>
<step>
<para>Put a copy ... <filename>bluefish-doctools/tools</filename></para>
</step>
</procedure>
```

If the explanation consists mainly in orders, you may want to use ordered list instead.

Using notes, tips, warnings

As we need them in places where there are not supposed to be, we use the following workaround:

```
<variablelist>
<varlistentry>
<term><emphasis>Warning</emphasis></term>
<listitem>
<para>You have to keep in mind...</para>
</listitem>
</varlistentry>
</variablelist>
```

Using links

To reference an external link, we use:

```
<ulink url="http://www.sourceforge.net">http://www.sourceforge.net</ulink>
```

Or:

```
<ulink url="http://xmlsoft.org/XSLT/">libxslt</ulink>
```

To reference an internal link (i.e. internal to the book), we use:

```
<xref linkend="getting-bluefish-updates"/>
```

This generates a linked text similar to "the section called ...". This is the preferred form, but it may not be always suitable; in this case, you can use:

```
<link linkend="getting-bluefish-updates">here</link>
```

To reference a chapter by number, we use:

```
Chapter <xref linkend="getting-bluefish" role="template:%n" />
```

Others tags

To highlight command line tools or small applications, we use:

```
<command>make</command>
```

To emphasize file or directory names, we use:

```
<filename>make</filename>
```

For user's instructions, use either:

```
<screen>$ make install</screen>
```

or:

```
Run the command <userinput>make</userinput>
```

Be aware that the former is shown alone on its proper line, while the latter is embedded within the line flow. If you use the `screen` tag, you should prepend either a `$` or a `#` followed by a space before the instruction, depending on how the command should be run, as non root for the former, as root for the latter. Moreover, with the `screen` command, we should check that the line is not too long, split it if needed, and add a backslash to indicate the splitting.

To embed chunk of code, we use:

```
<programlisting>
<![CDATA[
Run the command <userinput>make</userinput>]]>
</programlisting>
```

As a workaround a bug in **fop**, we use a special processing instruction to insert page breaks for PDF production. If the break is the same for A4 and USLetter format, the instruction is:

```
<?pagebreak?>
```

If it is only for A4 format, the instruction is:

```
<?pagebreaka4?>
```

Likewise for USLetter format only, it is:

```
<?pagebreakus?>
```

Similar processing instructions are used to insert line breaks for PDF production:

```
<?linebreak?>
<?linebreaka4?>
<?linebreakus?>
```

Recommendation

Do not use `simplesect` as it messes the table of contents.

Avoid to add blank lines or unnecessary white spaces in the files, it may break the files production and has the disadvantage to increase the files size.

A chapter should at least contains an `id`, a `title`, and either a `para` or section tag. Be aware that you cannot use an isolated `para` tag after a section.

All list items should use a `para` tag to embed their contents. If all of the items contents are very short, i.e. fit into one line, you may want to use the following attribute to suppress the additional line between items:

```
<itemizedlist spacing="compact">
```

The same applies to ordered lists.

Avoid contractions. Use *you will* instead of *you'll*.

Use the spell checker to correct any misspelling.

Producing the final files

You have the following options to make and remove the various files, which will be stored in the `bluefish-gtk2/doc` directory:

Options to make the Bluefish manual

make html
: Produces the book in html format.

make pdf-a4
: Produces the book in PDF format suitable for A4 paper.

make ps-a4
: Produces the book in PostScript format suitable for A4 paper.

make pspdf-a4
: Produces the PDF and PostScript versions for A4 paper.

make pdf-us
: Produces the book in PDF format suitable for USLetter paper.

make ps-us
: Produces the book in PostScript format suitable for USLetter paper.

make pspdf-us
: Produces the PDF and PostScript versions for USLetter paper.

make all
: Produces all versions of the book.

clean html
: Produces the book in html format.

clean pdf-a4
: Removes the book in PostScript format suitable for A4 paper.

clean ps-a4
: Removes the book in PDF format suitable for A4 paper.

clean pdfps-a4
: Removes the PDF and PostScript versions for A4 paper.

clean pdf-us
: Removes the book in PDF format suitable for USLetter paper.

clean ps-us
: Removes the book in PostScript format suitable for USLetter paper.

clean pspdf-us
: Removes the PDF and PostScript versions for USLetter paper.

clean all
: Removes all versions of the book.

make validate-all
: Validate the whole book.

If you want to create a new format, you will have to make a copy of the `titlepage-a4.xml` and `pdf-ps-a4.xsl` files in the `bluefish-doctools` directory, rename them according to the new format, and modify them.

Contact us

If you find errors in the manual, or just want to add more, please contact us. If you have questions on how to edit the manual that are not addressed in this appendix, you can always ask on the mailing list. Often, you can look to the chapter source to see how things are done.

Appendix D. GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

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

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program 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.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.