BBEdit 8

By John Gruber

All applications are software; but not all software is an application. The key difference is the primacy of the user interface in application development. To paraphrase Edward Tufte, the user interface *is* the application—whereas most non-application software doesn't even *have* a user interface.

From the perspective of a programmer, all software is just source code. In a broad sense, the source code for a UI-less background process such as the Apache web server very much resembles the source code for a GUI application. Developers' relationship with the software they produce is abstract, conceptual. When an artist wants to produce a circle, he draws a circle. When a programmer wants to produce a circle, he writes a few lines of source code. It is the ability to grasp these abstractions that separates programmers from non-programmers.

From the perspective of users, however, the importance of the user interface is profound. For users, the application is what they can see, click, and interact with. A user's relationship with an application is perceptual, sensual.

The raw capabilities of a particular application are, for most users, irrelevant; it's the *usability* that matters. Features which aren't presented via an intuitive, discoverable, usable interface might as well not even exist.

For something as inherently nerdy as a text editor, BBEdit is phenomenally popular. But its appeal is not universal: in addition to those who simply have no need for a serious text editor, or who simply prefer other editors, there are vocal contingents of BBEdit non-believers who profess outright bewilderment at BBEdit's decade-long dominance of the Mac text editor market.

There are two vectors for such bewilderment, both of which belie a genuine understanding of what it is about BBEdit's "interface" that makes it so beloved:

- One is the conflation of *aesthetics* with usability. The idea that the quality of an app's user interface is simply a measure of how good it looks; i.e., that the state of being "Mac-like" implies only adherence to the gestalt of Apple's recent-vintage Aqua-flavored visual whiz-bangery: gorgeous iconography, anti-aliased type, vibrant primary colors, and visual effects such as transparency, drop shadows, bezel edges, and smooth-gliding animated widgets.

  This is not to say that aesthetics are unimportant. To find something aesthetically pleasing is deeply satisfying in a left-brained way. But aesthetic appeal is but one aspect of user interface design, not the whole of it—and for a serious tool, not the most important aspect. Compare and contrast to, say, choosing an office chair. It's certainly nice to have a chair that looks good; but if you're going to be sitting in it 8 or more hours every day, ergonomics are much more important than aesthetics.

  I won't dispute that BBEdit's interface is relatively unadorned; but so while it's not *pretty*, it isn't trying to be. *Handsome* is perhaps more apt. The layout of windows, dialogs, and menus is meticulous and thoughtful. The point is not to impress you; the point is to enable to you get work done.

- The second is the belief that the user interface is worthy only of afterthought; that the real software is the underlying implementation, and that the user interface is but a wrapper. (Cf. April's "Ronco Spray-On Usability".)

  This view is generally expressed only by those who are extremely technically nerdy—i.e. the sort of guys who honestly see Mac OS X as *Unix with a Mac* GUI,

> rather than as an updated version of the Mac OS with Unix-like underpinnings. Their question to BBEdit advocates, invariably, goes something like: "What does BBEdit do that [*my favorite open source editor*] doesn't do?", where the variable is usually Emacs or vi/vim—or occasionally jEdit.
>
> The problem is that this is not the right question. Emacs, Vim, and their various derivatives are very fine editors, and inarguably offer many powerful features. They are also difficult to learn, cryptic to the uninitiated, and their human interfaces (such as they are) are unlike anything a Mac user would consider normal. Some would have you believe that such is the price of power—that *easy-to-use* and *powerful* are mutually exclusive.
>
> The appeal of BBEdit is in its balance of powerful text-editing features and an elegant, intuitive, and unabashedly *Macintosh*-style interface—and where by "interface" I don't mean in the sense of superficial cosmetic appeal, but in the deeper, interactive sense.
>
> It's also worth pointing out here that "easy-to-use" is probably not the right adjective. Once you have learned to use any software package, no matter how cryptic its user interface may be, it can be very "easy" to use. Hence it's true that one might find Vim easy to use if one takes the time to learn it and grow acclimated to its keystroke-driven interface. But that's a big "if". The Macintosh's main appeal is not that it is easy-to-use, but rather that it is easy-to-learn. The difference is significant, but often overlooked.

In short, the point of BBEdit is not just what it looks like or what it does, but how it feels. The challenge Bare Bones faces when releasing a major feature upgrade is not just to add to BBEdit's capabilities, but to do so in a Mac-like way.

Brief Aside on the Case for Using Emacs or Vim

Furthermore, it's also worth pointing out that, ironically, the most compelling case for adopting Emacs or Vim as one's primary editor is not their feature sets, but their *interfaces*. Specifically, the fact that they run *everywhere* in almost exactly the same way. Any system with a Unix-like shell can run these editors, and as of 2004, every significant desktop operating system either ships with a Unix-like shell or has one available as a free download. Thus, if you master Emacs or Vim, you can rightly feel quite confident that you'll be able to use it on every computer you use for the rest of your life.

If by circumstance or choice you use both a Mac and one or more non-Macs, it's an appealing thought to be able to use the same editor on every system—using the same keystrokes, shortcuts, and customizations.

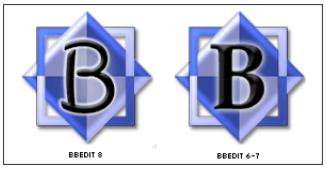
Text editing habits run deep. Mac users forced to switch to Windows regularly plead with Bare Bones to port BBEdit to Windows. Windows users who switch to the Mac regularly badger Bare Bones to make BBEdit more like their favorite editor from Windows (e.g. UltraEdit or TextPad). And, I'm guessing, after giving up on a BBEdit port to Windows, former BBEdit users start badgering for the BBEdit-ification of UltraEdit and TextPad.

Emacs and Vim users, on the other hand, just copy their configuration files from one machine to another—be it Windows, Linux, or Mac—and they're right at home.

BBEdit 8.0 First Impressions

Which brings us to BBEdit 8.0, the latest version of Bare Bones Software's flagship text editor, released with much fanfare two weeks ago. Before we proceed, let's just get it out of the way and link to my standard I-used-to-work-for-Bare-Bones-Software disclosure.

The first thing you'll notice is that BBEdit 8 sports a new application icon. The stately capital B set in bold Palatino—a hallmark of BBEdit's iconography since its public debut 12 years ago—has been replaced by something, well, a bit jauntier:



Reaction from the design community has not been kind:

- Jon Hicks:

  > With the release of BBEdit 8.0, has come the most inexplicable new feature. The application icon has been updated with the most grotesque typography, looking like a Comic Sans derivative.

  (Hicks whipped up a replacement icon, which, whatever you think of the new BBEdit 8 icon, is far worse—not because it's unappealing, but because it's un-BBEdit-y.)

- Andrei Herasimchuk:

  > But what the hell is up with the new BBEdit 8.0 icon? That has to be the worst looking "B" I've seen in some time. It almost looks like a "B" from Comic Sans! And we all know how catastrophic that is.

Suffice it to say the new icon is the most unpopular aspect of version 8. On the other hand, if the icon is the most-complained-about change, it speaks well for the application itself.

The most essential new aspect of BBEdit 8 isn't what it looks like, or even what it does, but what it *doesn't* do—which is run on Mac OS 9. In fact, not only does BBEdit 8 only run on Mac OS X, it requires 10.3.5, due to several bugs in earlier versions of Panther. (Having beta-tested BBEdit 8 under pre-10.3.5 builds of Panther, I can attest to this.)

This isn't just an idle milestone—BBEdit 8's Mac OS X-only nature informs nearly all of its major new features.

This is not to say the last few versions of BBEdit didn't take advantage of Mac OS X-only features. E.g., shell worksheets and the built-in Web Kit HTML preview, which for obvious reasons were only available when running BBEdit on Mac OS X.

Now that it only runs on Mac OS X, however, BBEdit 8 can take advantage of OS X-only features throughout the entire application, including UI widgets only available on OS X. Most of the additions and changes to BBEdit 8's user interface are tied to OS X-only controls and conventions.

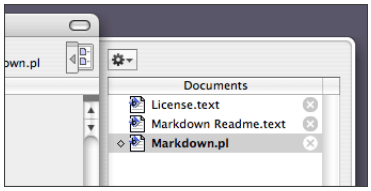
I see no need to provide a comprehensive overview of every new feature; Bare Bones' web site does a good job of that. And if that's not enough information for you, the full change notes are available: over 300 bullet items spanning 27 printed pages. Needless to say, by any measure, this is a significant update.

Multi-Document Interface

The most profound addition is the new multiple-document interface. Analogous to tabbed browsing with Safari, you can now open multiple documents within a single

BBEdit text window. A window's open documents are displayed in a drawer on the right side of the window.

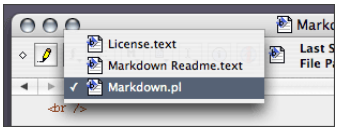
Thus, the multi-document interface is much more like OmniWeb's "tabbed" browsing than Safari's. And like OmniWeb, BBEdit 8 allows you to drag documents from one window to another. Clicking an icon in the status bar opens and closes the Documents drawer. (A nice touch: if the Documents drawer in the front window is closed, it will open automatically if you hover a dragged document on top of the drawer-toggling icon in the status bar.)



The multi-document interface is, of course, completely optional. If you're not interested, you can merrily continue using BBEdit using only the traditional one-document-per-window. Conversely, a few new document-related preferences allow you to configure BBEdit to *always* use the multi-document interface, putting new and opened documents in the frontmost window by default.

It's a radical departure from previous versions, where every text window represented one document, and every open document was in its own window. But it doesn't feel radical. It just works, exactly how you'd expect it to. You can ignore it, you can use it all the time, or, you can use it when you want. I've been using the feature since it debuted during beta testing, and while I tend to use the traditional one-document scheme most of the time (if for no other reason than a decade of habit), I've found the multi-document interface convenient for grouping related files together.

If you don't want to give up the screen real estate for the documents drawer, you can keep it closed and switch between open documents in the front window using the also-new Navigation Bar, which looks and works very much like the Navigation Bar in Xcode.



Unicode and Text Encoding Improvements

BBEdit 8 now uses ATSUI for text rendering. In all prior versions of BBEdit, the rendering engine was implemented using QuickDraw, a set of APIs dating back to the original Mac. This is an enormous under-the-hood change, text rendering being a core task of a text editor.

If you only use roman character sets, you likely won't notice that anything has changed—text editing and rendering looks and feels identical to that of BBEdit 7. Where the new ATSUI-powered engine shines is in the rendering of Unicode text. BBEdit now supports multiple languages—e.g. Chinese, Japanese, and English—all within the same document. If you don't need advanced Unicode support, BBEdit's new ATSUI-powered rendering engine works just as well as the old QuickDraw engine. If you do, you can now use BBEdit for tasks it previously couldn't handle. For a small subset of BBEdit users, this is one of the biggest improvements in the history of the app.

BBEdit's handling of text file encoding formats has also improved greatly. A new Reopen Using Encoding sub-menu in the File menu allows you to manually specify

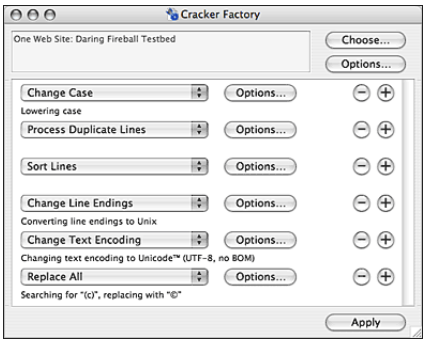
the text encoding for a given file. Thus, if you open a file saved using Windows Latin 1 encoding, but BBEdit displays it as Mac Roman, you can easily reopen the file with the proper encoding. With BBEdit 7, you needed to close the file, then reopen it using the Open dialog box, specifying the text encoding using a pop-up menu in the dialog. There were other ways to do this (Marc Liyanage's character set conversion Perl filters and the Midex BBEdit plug-in, for example), but nothing as easy or graceful as the new Reopen Using Encoding menu.

Text Factories

BBEdit's multi-file searching allows BBEdit to act as a production tool for batch processing many files at once. Sure, it's nothing a short Perl or Python script couldn't do—but if you think everyone can or should learn to program, you're missing the point.

But there's only so much a single search/replace can do. It's great if you need to something simple, like changing the copyright date across a few thousand HTML files. But if you need to string together multiple replacements, it's no longer a one-step process.

Enter "Text Factories". A Text Factory is a new type of document that contains a list of one or more actions. You can easily add or remove actions similar to the way you add/remove search criteria in the Finder's Find window: via '+' and '-' buttons next to each action. Text Factories seems conceptually similar to the Automator feature coming in Tiger.



Many of the commands in BBEdit's Text menu are available as Factory actions, as is the Search menu's Replace All command. You can change files' encoding and line-ending style (Mac/Unix/DOS). And for anything not covered by the built-in actions, you can also specify an AppleScript or shell script (Perl, Python, Ruby, bash, etc.) as an action item.

You can execute a Text Factory against a single document window, but the real power comes when batch processing files. If you regularly perform the same repetitive actions against multiple files, a Text Factory might be able to automate the entire process, without scripting.

And even if you're comfortable with scripting—whether with AppleScript or a shell language—Text Factories still might prove useful for batch processing. If you write your script to serve as a Text Factory action, the script only needs to be concerned with processing the text of a single file at a time, saving you from writing code to recurse folders and loop through, open, and save files—which I consider a minor pain in the ass even in Perl.

Multi-file Text Factory processing can also take advantage of the same file filtering mechanism used in BBEdit's multi-file searching, making it easy to limit Factory processing based on attributes such as file names, modification dates, and labels.

One obvious improvement for the future would be to allow Text Factories to be saved as "droplets"—small applications that you can drag files and folders to in the

Finder. Adobe Photoshop and ImageReady allow their actions to be saved as droplets, and it's a convenient way to invoke them when processing multiple files at once.

Multi-File Searching

Speaking of multi-file searching, BBEdit's mighty Find dialog has received a makeover. Turning on multi-file searching now opens a drawer on the left side of the dialog. The drawer contains a list of search targets: all open documents, recently-searched folders, and every HTML Web Site configured in BBEdit's preferences. It's better-organized and more convenient than the pop-up menus it replaces.

Plus, multi-file searches now run in their own thread, which means that searches can run in the background while you continue to work in BBEdit. Even though most searches I run take only a few seconds to run, it's a nice improvement when I do run a lengthy search.

Codeless Language Modules

I mentioned this feature previously, when I released my Apache Configuration language module. Out of all the new features in 8, this is probably the one that most deserves a response of "It's about time." Prior to version 8, language modules for BBEdit needed to be compiled plug-ins written in C/C++. The advantage to this is that BBEdit's compiled language modules are both powerful and very fast. The disadvantage is that most people aren't willing or able to create them on their own.

Most text editors follow the opposite approach, and *only* support languages modules in a plain text format. Now that BBEdit supports both, it should provide a best-of-both-worlds experience.

However, BBEdit 8.0's codeless language module mechanism currently has a few shortcomings. It's more than a little biased toward programming languages, with good support for things like functions, string delimiters, and keywords. What it's not so good for are markup languages. (Like, say, Markdown.) The current CLM mechanism has no concept of things such as tags, and it doesn't allow for using regex patterns to match language constructs.

Nor does it allow for embedding one language within another, the way BBEdit's built-in language module for HTML supports inline chunks of other languages, such as PHP, ASP, and CSS.

Plug-ins

Speaking of compiled plug-ins, BBEdit 8 now supports plug-ins compiled using Xcode (i.e. Mach-O binaries). Previously, BBEdit required plug-ins to be CFM binaries, which effectively meant they had to be compiled in CodeWarrior. This restriction pretty much eliminated hobbyist-level plug-in development, combined with the fact that many of the text-munging tasks tackled with plug-ins a decade ago are now much more easily written in Perl or Python. BBEdit 8 still supports old-style CFM plug-ins, with the sole restriction that it no longer supports plug-ins that aren't Unicode aware.

Also, a bunch of plug-ins that used to ship with BBEdit are now built-in commands in the Text menu: Add/Remove Line Numbers, Prefix/Suffix Lines, Sort Lines, Process Duplicate Lines, and Process Lines Containing. This, combined with the fact that other third-party plug-ins have been obsoleted by other new features in 8.0, has left me with zero plug-ins (other than language modules). Examples of obsoleted plug-ins: BBTidy (HTML Tidy is now available via BBEdit's Markup menu) and Midex (no longer necessary thanks to the aforementioned Reopen Using Encoding sub-menu). I had a

plug-in I wrote years ago for myself but never released, Stupefy Quotes, which has been obsoleted by the new Straighten Quotes command in the Text menu.

I'm be interested to see whether support for Xcode spurs a surge in plug-in and compiled language module development. I'm guessing no for plug-ins—Unix Filters have eaten their lunch—but yes for compiled language modules.

Support Files

By dropping Mac OS 9 support, BBEdit is now able to fully support OS-X-style conventions for supporting files:

- The application support folder is now named "BBEdit" instead of "BBEdit Support". More importantly, it now supports proper domain layering, by which I mean that you can put a "BBEdit" support folder in `/Library/Application Support/`, and it will be available to any users on the machine running BBEdit. But each user can also have their own BBEdit support folder in the Application Support folder in their user Library folder, and the items therein will be used in addition to those from the top-level Library folder.

- Custom keystroke shortcuts assigned to items such as AppleScripts, Glossary items, and Unix filters are no longer stored in the resource forks of each individual file. Instead, custom keystrokes are stored separately from the files, in a similar way as the custom menu key shortcuts for regular menu items. One reason for this is that users might not have sufficient privileges to write to the files in the top-level Library folder. Another is that it makes it possible for BBEdit to do a better job identifying and preventing keystroke conflicts.

- BBEdit now uses OS X's built-in preferences system. Thus there's now a '`com.barebones.bbedit.plist`' file containing your preference data, and, if necessary, you can set or check preference values using the `defaults` command-line tool—or you can hack a copy of plist file using BBEdit itself. There's also a '`com.barebones.bbedit.PreferenceData`' folder (named as such so that it sorts next to the regular preferences file in the Finder), containing user preferences that aren't saved in the regular prefs file—including saved grep patterns, file filters, and S/FTP bookmarks.

Document State

Document state is text-file-specific metadata; for BBEdit, this includes things like the position and size of the window, the position of the scrollbar, the current text selection range, and the text encoding used to save the file. In all previous versions of BBEdit, state data was stored in the resource fork of each document. This technique was pioneered, I believe, by MPW (and BBEdit's state resources were a superset of MPW's).

This scheme was quite clever, and worked exceedingly well in the old Mac OS world. The only thing *essential* is the actual text of the document, which was stored in the data fork. By stashing state info in the resource fork, you could preserve it by passing it from one resource-fork-savvy medium to another. Send it to someone on Windows or Unix, and the state data would be lost, but the actual text in the data fork would be unaffected.

In the new world of Mac OS X, however, storing state data in resource forks has significant downsides. For one thing, resource forks don't survive the roundtrip through version control systems such as CVS or Subversion. It also causes problems with files are stored on file systems that don't support multiple forks, such as WebDAV or UFS.

For those reasons and others, BBEdit 8 has switched to an entirely new system for storing document state. Instead of storing state data inside each file, you get a single preference file that contains the state info for every file you edit. It's stored in a new file inside com.barebones.bbedit.PreferenceData: "Document State.plist".

This solves all of the aforementioned problems with resource fork state info—you can commit a file into CVS, check out a new revision, and your state settings for that document will be preserved. If you work with multiple collaborators on the same set of files in a CVS repository, each of you can maintain your own state settings for each file.

This is not to say there aren't downsides to the new system. The new centralized state database is fragile—it references individual files by, of all things, a hardcoded pathname. Thus if you rename or move a file, BBEdit loses track of its state. Move it back and the state settings come back.

This is clearly a trade-off. For anyone who works in a cross-platform environment or depends upon non-resource-fork-savvy tools, the new system clearly works better. However, for anyone for whom resource fork state storage did not pose a problem, the new system may not work as well, particularly for anyone accustomed to state persisting when you rename, move, or copy the file.

Other Miscellaneous Improvements Based on Mac OS X Technology

- BBEdit now optionally supports running pages through the Apache web server on your own machine before previewing them. Thus you can preview PHP-powered pages, and the preview will show the actual PHP-rendered output. There's a world-class web server on every copy of Mac OS X, why *not* use it?

- BBEdit now uses Mac OS X's built-in spelling checker. The advantage isn't that the system's spelling checker is particular great, but that BBEdit's old built-in checker was rather clumsy and annoyingly modal. BBEdit doesn't yet support check-as-you-type, but it does use the standard Spelling palette, and nicely underlines misspelled words in the document window itself. (The system's Spelling palette inexplicably doesn't have a Correct All button, and so neither does BBEdit. I hope this gets fixed, either by Apple in 10.4 or by Bare Bones in a future update to BBEdit.)

- BBEdit now uses the system's Fonts panel for choosing the display font and tab width of the current document.

Other Details Worth Mentioning

- In addition to the `bbedit` command-line tool that started shipping with BBEdit 6.5, BBEdit introduces a `bbdiff` tool for comparing the contents of files and folders. This completely obsoletes the Perl script of the same name I slapped together two years ago. Bare Bones' new tool does the same things as my script, plus more.

- BBEdit 8 introduces several new text color preferences, including an option to subtly highlight the current line. It also allows you to specify custom colors for the selection highlight, independent of the system-wide setting set in System Prefs—this is useful if you decide to use a dark background with light text.

- When BBEdit's Balance While Typing option is on, you get a brief notification whenever you type an unbalanced closing parenthesis, bracket, or smart quote. In

the old Mac OS, this notification took the form of briefly flashing the menu bar at the top of the screen. Easy to notice, but subtle enough so as not to be annoying. On Mac OS X, however, the API call that flashed the menu bar under Mac OS 9 and earlier does something different: it briefly flashes the entire screen. Definitely noticeable, and in many BBEdit users' opinion, *too* noticeable.

BBEdit 8 offers a much more OS-X-like notification: a transparent status window and fades in and out.(similar the ones the system displays when you change the volume or display brightness).

In an exceedingly nice touch, if the unmatched character is itself a smart quote, BBEdit uses straight quotes to delimit the character in the notification message.

Pricing and Discounts

The price for a new license has been raised to $199, but through the end of October, BBEdit 8 remains available for the old price of $179. Upgrades from version 7 are $49, and upgrades from *any* previous version are only $59. Plus, BBEdit's long-standing "cross-grade" discount remains: owners of Dreamweaver, GoLive, and BBEdit Lite can purchase BBEdit for only $129. (Yes, BBEdit Lite, which was available free of charge. No I don't have something in my eye—I'm *wink*ing.)