# WASTE 2.1 Change History

## New in WASTE 2.1b1 (August 2002)

WASTE 2.1 goes beta!

A lot of fixes in this version address problems introduced by the new highlighting algorithm that made its debut in the previous build:

• `WEUndo`/`WERedo` could sometimes leave stray highlighting.

• Another instance of stray highlighting would occur when option+dragging some text from one line to another within the same document. Thanks to Jérôme Seydoux.

• On classic Mac OS systems with the **Kaleidoscope** installed, the non-Carbon builds of WASTE would use a white highlight color, effectively making the highlight invisible on a white background. While I think this is a Kaleidoscope bug, this version of WASTE includes a workaround. Thanks to Hiroki Hisaura.

• WASTE would fail to draw a visible highlight on patterned backgrounds as well. Thanks to Alessandro Volz.

• WASTE now has better support for patterned backgrounds, like the theme brushes for dialog backgrounds used on OS X. First of all, patterned background are now compatible with the offscreen drawing feature (the background pattern, if any, is copied to the gworld used internally). Secondly, WASTE will now avoid using `ScrollRect` for scrolling text if it detects the presence of a pattern — this allows text to move while the background remains fixed.

• Following a suggestion by Manfred Schubert, I added some new accessors (`WEGetObjectBaseline`, `WESetObjectBaseline` and `WESetObjectPlacement`) that allow getting and setting the **intrinsic baseline** of an embedded object (for a discussion of intrinsic baselines, see below: New in WASTE 2.1a5).

• A new `WEGetMaxLineWidth` API returns the pixel width of the widest line. Thanks to Mark Alldritt.

The result is the actual maximum right after a call to `WECalText`; later, the value returned by `WEGetMaxLineWidth` is only an *upper bound* on the width of the widest line: the actual maximum may be less than the value returned. If you need the *exact maximum* at any given time, loop over the lines calling `WEGetLineWidth` repeatedly. `WEGetMaxLineWidth` can be useful for setting up the horizontal scroll bar when you have a really wide destination rectangle.

• Fixed a problem in `WEAdjustCursor`, likely introduced in version 2.1a7, where WASTE would not turn the cursor to an I-beam if the mouse was past the right edge of the destination rectangle, but still within the view rectangle.

• Fixed problems with compiling the source code with `WASTE_DRAG_AND_DROP` set to zero. Thanks to David Rydh and Linda Peting.

## New in WASTE 2.1a9 (June 2002)

• This version of WASTE comes with a completely rewritten algorithm for drawing

**selection highlights**. Previously, we would just swap the background color with the highlight color in the selection region. This simple approach has the advantage that inverting the selection region a second time removes the highlight — no need to redraw the text. This used to work perfectly in the early days when you mostly had black text on a white background, but doesn't look good with anti-aliased text (and especially with Quartz anti-aliased text). For a more in-depth discussion of this problem, see:

<http://developer.apple.com/technotes/tn/tn1149.html#Section3>

The new approach involves erasing the selection region to the highlight color, then drawing the text over in `srcOr` mode. The result is much better looking selection highlights.

Also, on Mac OS X WASTE will now mark selections in inactive windows with a solid highlight drawn in a lighter shade of the system-defined highlight color, as per the Aqua Human Interface Guidelines. Thanks to John C. Daub.

• The WASTE Demo application now takes advantage of **Quartz text rendering** on Mac OS 10.1.5 and newer.

• Fixed a problem where the "drag caret" (the caret shown as a destination feedback while tracking a drag) wouldn't blink on Mac OS X. Thanks to Kuniaki Nakamitsu.

Also, on OS X the text auto-scrolls as expected during drag-tracking.

• A new feature flag (`weFNoAutoTabForHangingIndent`) prevents WASTE from adding an automatic tab stop to paragraphs with a hanging indent (i.e., where the first-line indent is a negative value). While the automatic tab stop makes it easier to create numbered or bulleted lists, there are situations where it just gets in the way.

Also, fixed a related bug where the automatic tab stop would only be added to paragraphs with no user-defined tab stops.

• WASTE will now prefer `FMGetFontFamilyFromName` over `GetFNum` and `FMGetFontFamilyName` over `GetFontName` where the newer Font Manager APIs are available.

# New in WASTE 2.1a8 (May 2002)

• A new **undo notification hook** allows your application to be notified whenever an undo-related event occurs. You declare your callback like this:

```
pascal OSErr UndoProc
    (
        SInt16                          inUndoEvent,
        WEActionReference               inAction
    ) ;
```

The `inUndoEvent` parameter describes what is happening and can take up one of three possible values:

`weUndoNewAction`

A new action is being pushed onto the internal undo (or redo) stack. This happens when some undoable action (e.g., typing, inline input, pasting) is occurring and undo is enabled. Also, this event is generated when a non-empty action sequence is closed by a call to `WEEndAction`.

`weUndoBeginSequence`

A new action sequence is being opened. This happens when `WEBeginAction` is called, either explicitly or internally by WASTE (e.g., during a drag-move, which is actually two actions being treated as one).

`weUndoNewSubSequence`

A new action is being added to the open action sequence. This only happens when an undoable action occurs between `WEBeginAction` and `WEEndAction`.

The `inAction` parameter is an opaque reference to the action involved. Some new accessors allow you to extract relevant information from a `WEActionReference`, like its action kind (`WEGetActionKind`), whether it's an undo or a redo action (`WEActionIsRedo`), and the WASTE instance associated with the action (`WEGetActionOwner`).

• Added a new `WECopyToScrap` API that gives you more control on the way WASTE performs the familiar copy operation.

```
pascal OSStatus WECopyToScrap
    (
        ScrapRef                        inScrap,
        const ScrapFlavorType *         inScrapFlavors,
        OptionBits                      inCopyOptions,
        WEReference                     inWE
    ) ;
```

The `inScrap` parameter allows you to specify a scrap other than the global clipboard or pasteboard. You can pass `NULL` in this parameter to refer to the global scrap. This parameter is not significant in classic (non-Carbon) builds and is always ignored.

The `inScrapFlavors` parameter allows you to specify an optional, zero-terminated array of flavors to be copied. Pass `NULL` to let WASTE choose the appropriate flavors.

A parallel `WEPasteFromScrap` API lets you perform a paste operation from a specified scrap:

```
pascal OSStatus WEPasteFromScrap
    (
        ScrapRef                        inScrap,
        OptionBits                      inPasteOptions,
        WEReference                     inWE
    ) ;
```

• You can now install an optional **busy callback** that is invoked by WASTE at specific intervals during time-consuming operations.

• Added a `WEGetLineWidth` accessor that returns the pixel width of the specified line.

• An internal API used to highlight text ranges in response to requests by Text Services Manager input methods has been exposed:

```
pascal OSStatus WESetTSMHilite
    (
        SInt32                          inRangeStart,
        SInt32                          inRangeEnd,
        WETSMHiliteStyle                inTSMHiliteStyle,
        WEReference                     inWE
    ) ;
```

Valid values for `inTSMHiliteStyle` parameter are `kRawText`, `kSelectedRawText`, `kConvertedText` and `kSelectedConvertedText` (defined in `<AERegistry.h>`), plus more values defined in WASTE.h (`weRedWigglyUnderline`, etc.) that may be handy for implementing MS Word-style wiggly underlines for misspelled words.

• **Single-click selection of lines**: When the `weFLeftMarginClick` feature flag is enabled, a single click to the immediate left of a line selects the whole line. A double click selects the corresponding paragraph. Left margin clicks can be combined with the shift modifier to extend the selection in the usual fashion. If the `weFLeftMarginClick` flag is on, and `WEAdjustCursor` is called for a point to the left of the destination rectangle, the cursor is changed to a northeast-pointing arrow.

• **BBEdit-style combos for backspace and delete**: option-backspace deletes the word to the left of the cursor, command-backspace deletes to the beginning of the line and command-option-backspace deletes to the beginning of the document.

• Added some support for **RTFD packages** (RTF documents with external attachments). In particular, `WELoad` will recognize such packages when asked to guess the file type, and the RTF parser will understand the proprietary NeXTStep keyword used to embed graphics (only TIFF graphics is handled at this time). Also, `WESave` can now write RTFD packages, saving inline graphics in a way that can be understood by Cocoa applications.

• The RTF parser now does a better job at matching font names. In particular, the parser now understands Postscript font names found in RTF files produced by Cocoa applications.

• Fixed a problem where WASTE would assume the wrong text encoding for ranges set in some Mac OS X fonts like Hiragino, which have a Quickdraw font family number outside the (now obsolete) Script Manager range for the corresponding script. This problem caused WASTE to return bogus Unicode text for such ranges.

• Fixed a subtle cosmetic problem with drawing `weTagBorderStyleDotted` borders. The gray pattern we used was not aligned to the pen position.

• WASTE Demo: added support for mouse wheels, contextual menus and Mac OS X Services.

# New in WASTE 2.1a7 (March 2002)

• **Line spacing modes**.

Line spacing modes allow more control on the way line heights are calculated. Previously, the `weTagLineSpacing` tag was a `Fixed` value functioning as a multiplier for the height of lines in a paragraph. A value of `0.0` (the default) meant natural line height (make the line as tall as the tallest character in it), `1.0` meant double line height and, more generally, a non-negative value `X` meant make the line `(1 + X)*N` pixels tall, where `N` is the natural line height.

Starting from WASTE 2.1a7, the attribute getters and setters accept a `WELineSpacing` structure in their value parameters, defined as follows:

```
typedef struct WELineSpacing
{
    WESelector      mode ;
    Fixed           value ;
} WELineSpacing ;
```

The `mode` field can be one of `weTagLineSpacingRelative`,

weTagLineSpacingAbsolute or weTagLineSpacingAtLeast. The relative mode instructs WASTE to treat the value field as it always had in the past, i.e., as the relative multiplier previously described. But when the other two modes are in effect, the value field is interpreted as an absolute (exact) pixel height, expressed in Fixed points as usual.

Futhermore, when the "at least" mode is in effect, WASTE will use the value field or the natural line height, whichever is greater, whereas when the "absolute" mode is specified, WASTE will use the value field even if it's less than the natural line height, possibly cropping some glyphs in the process.

The attribute getters and setters still accept Fixed values for the weTagLineSpacing tag, for compatibility with older versions of WASTE.

• The delta between the natural height of a line and the height specified by the line spacing tag is now distributed proportionally between the ascent and descent portions of a line (previously, it was added to the descent portion only).

• The RTF I/O module now recognizes and writes a number of paragraph-level keywords controlling pagination and hyphenation.

• Modified the implementation of some deprecated APIs (WEGetParaInfo, WEGetIndParaInfo, WESetRuler and WEContinuousRuler) so that they cannot be used to inspect or set any attributes besides those explicitly documented in the public headers.

• Fixed a bug where the sequence:

```
WEBeginAction ( we ) ;
WEFeatureFlag ( weFUndo, weBitClear, we ) ;
WEKey ( key, modifiers, we ) ;
```

caused a crash. Thanks to Olivier Destrebecq for reporting this.

• Incorporated a fix submitted by Matsumoto Satoshi for an undo-related problem that manifests itself with the ATOK input method for Japanese.

• Fixed a bug reported by Yoshio Hirose where the caret would not be positioned correctly in lines with tab characters set in non-Roman fonts.

• Fixed a bug in WESetAttributes reported by Peter Hoerster: weTagSubscript was broken.

• Fixed a potentially crashing bug in the RTF parser where loading a document containing PNG- or JPEG-encoded pictures could cause a crash with QuickTime 2.5. Thanks to Tomasz Kukielka for pointing out the problem.

• Fixed a bug where WELoad was ignoring the rangeStart/rangeEnd parameters when loading RTF files. Thanks to Fabio Peri.

• Fixed a bug in WASTELib where the host application would be abruptly terminated if a C++ exception was thrown by the RTF I/O module. This would typically occur when loading a corrupt RTF file.

• The "Apple Chancery" and "Zapf Chancery" are now classified as \fscript fonts rather than \fdecor fonts by the RTF writer.

• WASTE.h: added experimental support for framework-style includes.

• WASTE Demo: the minimum system requirements are now Mac OS 9.1 + CarbonLib 1.5 for Classic and Mac OS X 10.1 for X.

# New in WASTE 2.1a6 (January 2002)

• **Tabs**. Each paragraph can now have its own set of user-defined tabs. Up to 20 tab stops per paragraph can be defined.

Default tabs are used in paragraphs that don't have user-defined tabs, or when a tab character is encountered past the rightmost user-defined tab on a line. Default tabs are placed automatically at integer multiples of the **default tab width**, a document-wide value that can be gotten and set using suitable accessors (`WEGetDefaultTabWidth` and `WESetDefaultTabWidth`).

An automatic tab stop is added at the left indent position if there is a hanging indent, i.e., if the first line indent is negative.

Tab indents are expressed in Fixed points from the left edge of the destination rectangle. Tabs can have different alignments (left, center, right or decimal), although currently only left-aligned tabs are rendered correctly. Tabs can also have leaders, i.e., repeating characters used to fill the blank space.

Use `WESetAttributes` or `WESetOneAttribute` with the `weTagTabList` selector to apply a tab set to a range.

• The built-in 'new' handler for pictures now detects a couple of special picture comments used by equation editors and other programs to specify the equation baseline. To see this code in action, try creating a simple equation in Graphing Calculator, copy it and paste it into the WASTE Demo.

• Implemented the all caps, all lowercase and hidden styles.

• Added the `WEGetLineAscent` accessor.

• Numerous fixes and enhancements to the RTF I/O module.

• The WASTE static libraries no longer need the external symbols `__shl2i` and `__shr2u` (defined in the Metrowerks runtime libraries). Thanks to Theo Vosse.

• Fixed a potentially crashing bug occurring when activating, deactivating and disposing of a WASTE instance with the `weFAutoIdle` feature enabled. Thanks to Jérôme Seydoux of the PowerMail team.

# New in WASTE 2.1a5 (January 2002)

• It is now possible for a 'new' object handler to specify the **intrinsic baseline** of an object, i.e., a vertical shift to be applied to the bounding rectangle of the object when it's drawn. This amount is added to any vertical shift applied with `WESetAttributes`. The first parameter to the new handler, although declared as a `Point *` for compatibility with older versions of WASTE, is now actually a pointer to a `WEObjectPlacement` structure which includes an `objectBaseline` field. Setting this field to a positive value raises the object above the text baseline, while setting it to a negative amount lowers the object below the baseline.

• `WESave` and `WELoad` would check for the wrong Gestalt bit (`gestaltFSSupportsHFSPlusVols` instead of `gestaltHasHFSPlusAPIs`) and could crash if called on pre-Mac OS 9 system software.

• `WESetAttributes` would incorrectly reject negative tab indents.

# New in WASTE 2.1a4 (December 2001)

Plenty of new features and bug fixes in this release, and more to come...

• New filing APIs allow you to store the contents a single WASTE instance (or a subrange of it) into a file, in one of several file formats, and load them back. These APIs are still in flux and have several known problems that will be addressed in future releases. Please bear with me!

```
pascal OSStatus WESave
  (
    SInt32                inRangeStart,
    SInt32                inRangeEnd,
    const FSRef *         inFileRef,
    OSType                inFileFormat,
    TextEncoding          inTextEncoding,
    OptionBits            inSaveOptions,
    WEReference           inWE
  );
```

`WESave` saves the text range specified by `inRangeStart` and `inRangeEnd` to `inFileRef`, in the format indicated by `inFileFormat`. Currently, the following file formats are supported: `kTypeText ('TEXT')`, `kTypeUnicodeText ('utxt')` and `kTypeRTF ('RTF ')`.

`kTypeText` specifies a classic Macintosh text files (using traditional Mac OS encodings). All formatting attributes and embedded objects will be lost unless you specify `weSaveAddResources` in the `inSaveOption` parameter, which causes WASTE to add formatting scraps to the resource fork of the file. Specify `weSaveCompatibilityResources` in addition to `weSaveAddResources` to include formatting scraps suitable for applications based on TextEdit and WASTE 1.x. Please keep in mind that the text encoding(s) used in a classic Macintosh text file cannot be reliably inferred unless style information is present.

`kTypeUnicodeText` specifies a UTF-16 encoded text file. By default, UTF-16 text is output in big-endian byte order. Specify `weSaveLittleEndian` in `inSaveOptions` for little-endian (Intel) byte order. A byte-order mark is always automatically prepended at the beginning of the file.

`kTypeRTF` specifies a Rich Text Format file. The flavor of RTF used by WASTE (based on the Microsoft RTF 1.5 specification) supports the full range of character-level and paragraph-level formatting attributes currently understood by WASTE, plus a number of other attributes not currently used by WASTE but useful for roundtripping. It also supports embedded pictures, but not other kinds of embedded objects. This flavor of RTF is currently unsuitable for saving non-Roman text. The RTF option is currently only available in the WASTELib dynamic library, but not in the static libraries.

The `inTextEncoding` parameter is reserved for future use: pass `kTextEncodingUnkown` for the time being.

```
pascal OSStatus WELoad
  (
    SInt32                inRangeStart,
    SInt32                inRangeEnd,
    const FSRef *         inFileRef,
    OSType *              ioFileFormat,
    TextEncoding *        ioTextEncoding,
```

```
    OptionBits*          ioLoadOptions,
    WEReference          inWE
  );
```

`WELoad` loads the contents of a file back into a WASTE instance, replacing the specified text range. The file format here is an input/output parameter. You can pass `typeWildCard` (`'****'`) on input if you want WASTE to figure out the correct file format for you; on output, `WELoad` will returned the detected format. Or you can specify a real file type on input to force `WELoad` to interpret the contents of the file in a certain way; for instance, this is useful if you want to open an RTF file as a plain text file to edit the RTF tags manually.

The `ioTextEncoding` and `ioLoadOptions` parameters are reserved for future use: pass `NULL` for the time being.

`WESave` and `WELoad` require Mac OS 9.1 or later. I'm investigating a versions of these APIs that will take `FSSpec`'s and work on older versions of the system software.

• A new, Carbon-only `WEProcessHICommand` is meant to streamline code for handling menus and toolbars.

```
pascal OSStatus WEProcessHICommand
  (
    const HICommand *    inHICommand,
    WEReference          inWE
  );
```

It will handle most common editing commands such as undo, redo, cut, copy, paste, clear and select all.

`WEProcessHICommand` can also process menu commands that set text attributes, like those generated by a Font menu or an Alignment menu, provided you have set up your menus by associating suitable properties to the menu items. For example, if you have a Text Color menu that you want WASTE to handle, you should associate an `RGBColor` property to each menu item using `SetMenuItemProperty`. The creator tag of the property must be `kWASTECreator` (`'OEDE'`), and the property tag must match the WASTE attribute tag. Finally, the command ID of the menu item must also match the WASTE tag of the attribute you want to set.

• WASTE now features built-in object handlers for the most common object types (currently, just pictures and sounds). A new API allows you to install the built-in handlers for a given type:

```
pascal OSErr WEInstallStandardObjectHandlers
  (
    FlavorType           inFlavorType,
    WEReference          inWE
  );
```

Passing `typeWildCard` (`'****'`) in `inFlavorType` installs handlers for all supported object flavors. A corresponding `WERemoveStandardObjectHandlers` API uninstalls the handlers.

• A new Carbon-only feature flag, `weFAutoIdle`, instructs WASTE to take care of caret blinking and other idle-time chores automatically. If you specify this flag in `WENew` (or enable it using `WEFeatureFlag` before the first call to `WEActivate`), you don't need to call `WEIdle` periodically as WASTE installs an event loop timer on your behalf.

• Added more flexible APIs for attaching user-defined data to a given WASTE instance:

WEGetProperty, WESetProperty and WERemoveProperty. Unlike the existing WEGetUserInfo, WESetUserInfo and WERemoveUserInfo, these APIs allow you to attach chunks of data of any size (not just 4 bytes), and identify the chunks by tag **and** creator.

These new APIs are implemented through the Collection Manager, so they are only guaranteed to be available in Carbon or in classic Mac OS 8.5 and newer. If you attempt to use them on a system that doesn't have the Collection Manager, you'll get an unimpErr result code.

In Carbon builds of WASTE, the old UserInfo APIs are now implemented on top of the Property calls. This means that client code expecting a specific error code (-50, or weUndefinedSelectorErr) when the specified user info cannot be found will have to be revised to look for -5751, or collectionItemNotFoundErr.

Added similar APIs for embedded objects: WEGetObjectProperty, WESetObjectProperty and WERemoveObjectProperty.

• A number of formatting attributes defined in the RTF specification and used in Microsoft Word and many other word processors can now be stored internally by WASTE, including hidden, all caps, small caps, embossed, engraved, double-strikethrough, and several underline styles (word, double, dotted, dashed, thick, wave, etc.). Also, WASTE can now remember tab stops, tab alignments and tab leaders associated with each paragraph.

Although none of the above attributes are currently used in the rendering code, the RTF I/O module takes advantage of this capability to provide better roundtripping of RTF files.

• The RTF I/O module will also preserve several pieces of document-wide information often found in RTF documents: title, subject, author, manager, company, category, keywords, comments, operator and the base address for hyperlinks.

• Added support for language tags. Language tags are character-level attributes that specifies the language (and optionally, other locale-related hints, such as the geographic region) of a text range. They're not currently used by WASTE (except for roundtripping of language information in RTF files), but they're suitable for a number of future uses, such as spell-checking, thesaurus look-up, hyphenation and locale-savvy smart quotes.

A language tag is basically a lowercase ISO-639 two- (or three-) letter language code, optionally followed by an underscore and by an uppercase ISO-3166 country code. For example, the following code marks the selection range as US-English:

```
err = WESetOneAttribute ( kCurrentSelection,
  kCurrentSelection, weTagLanguage, "en_US", 5, myWE ) ;
```

An empty tag ("") means that the language is unknown or unspecified.

The current implementation of language tags relies on the Mac Locale APIs introduced in Mac OS 8.6, so it's non-functional on earlier versions of the system software.

• A new read-only selector for WEGetAttributes, weTagRunDirection, returns true if the character at the specified offset belongs to a right-to-left script system. This new selector is a replacement for the old, seldom-used WEGetRunDirection API, which is now deprecated.

• A new WEGetSelectionAnchor API returns the offset of the selection anchor, i.e., the endpoint that stays put when you shift-extend the selection.

• You can now pass the wePutAddToTypingSequence option to WEPut even if you're

not in the middle of a typing sequence (but you're adding new text at the insertion point). In this case, `WEPut` will create a new undoable action that behaves like a typing sequence. This allows several `WEPut` calls in a row, all appending text at the insertion point, to be treated as a single, typing-like, undoable action.

• The sequence `WEBeginAction`/`WEEndAction`/`WEUndo` (with no intervening editing between `WEBeginAction` and `WEEndAction`) would crash WASTE. Now, closing an empty action sequence with `WEEndAction` will remove the empty action from the undo stack. In other words, calling `WEBeginAction` immediately followed by `WEEndAction` has no effect. Thanks to Jerry Åman.

• `WEKey` no longer bumps the modification count if you hit backspace when the insertion point is at the very beginning of the text, or if you hit forward delete when the insertion point is at the very end. Thanks to Chris Eplett.

• Fixed a long-standing, elusive bug in the paragraph formatting code. To see what this bug did to earlier versions, try this: insert two paragraphs into a blank window, change some paragraph-level attribute of the second paragraph (e.g., make it centered). Then select the first paragraph, including the trailing carriage return, and delete it. In most cases, the second paragraph would incorrectly take up the formatting of the one just removed. The weird thing was, this wouldn't happen for any pair of paragraphs following the first one. Thanks to the invaluable Kuniaki Nakamitsu.

• Fixed a problem in the RTF parser: MS Word (and possibly other programs as well) use the `\highlight0` idiom to remove the highlighting altogether, while WASTE would set the highlight color to black, making the text illegible. Thanks to Casey Gorsuch.

• Converted this document (the Change History) to RTF format.

# New in WASTE 2.1a3 (October 2001)

• `WENew` could potentially crash in low-memory situations. Fixed.

• `WEGetIndUndoInfo` would likely crash if passed zero in `inUndoLevel`. Fixed.

• `WEBeginAction` and `WEEndAction` were broken in version 2.1a2. Hopefully fixed. Thanks to David Rydh.

• Paragraph bottom borders are now drawn even when the paragraph `spaceAfter` value is less than 2 pt. Thanks to Tom Bender and David Rydh.

• Incorporated tentative fix for the "disappearing hilite" bug noted when undoing/redoing some action and when the current selection and the restored selection ranges live on different lines. Thanks to Kuniaki Nakamitsu and David Rydh.

• `WEKey` now handles page keys (page up, page down, home and end).

• Source code: WASTE wouldn't link if compiled with the `WASTE_DRAG_AND_DROP` switch turned off.

• Source code: eliminated warnings about missing braces in `struct` initializers and missing parameter names given by Project Builder.

# New in WASTE Demo 2.1a3 (September 2001)

This is a major overhaul of the demo application shipped with public WASTE distributions from version 1.x to 2.0b3. The new demo is Carbon-only, requires Mac OS

9.1 or newer and is simpler than the old demos. Among the new features:

• Event dispatching is now based on Carbon events. This means less code and (hopefully) better CPU utilization. It will mean even less code as soon as WASTE is updated to better support the Carbon event model.

• `FSSpec`'s have been replaced by `FSRef`'s everywhere. As a side effect of this change, we now support long (> 32 characters) file names and file names containing arbitrary Unicode characters.

• Navigation dialogs now use (actually, require) Navigation Services 3.0 or newer. All navigation dialogs are either window-modal or modeless on OS X.

• Document windows support live resizing.

• Several seldom-used menus have been removed for the sake of simplicity.

## New in WASTE 2.1a2 (August 2001)

• The source code has been cleaned up so that it compiles with the latest Universal Headers (3.4), CodeWarrior Pro 7 and Project Builder.

• A Mach-O static library (WASTE.a) is now included in the public distribution.

• The WASTE Demo has been repackaged as a hybrid CFM / Mach-O bundle that can run under Mac OS 8.6 through 10.1.

• The Carbon builds of WASTELib will now call Internet Config directly rather than through the Component Manager interface, as the latter is not implemented in Mac OS X. In other words, command-clicking URLs now works again on Mac OS X.

• Various private data structures are now malloc-based rather than handle-based. In particular, storage for WASTE instances, action records (used in undo / redo stacks), print sessions and embedded object descriptors is now obtained through `malloc` calls. This eliminates a lot of extra pointer dereferences and locking / unlocking of handles. Old-style Memory Manager handles are currently still used for growable blocks.

This is a purely internal change: an implementation detail you should not be concerned with unless your code incorrectly assumes *opaque reference types* like `WEReference` and `WEPrintSession` are actual Memory Manager handles.

• The Carbon version of WASTELib will use `TrackMouseLocationWithOptions`, if available, instead of a `GetMouse`/`WaitMouseUp` combination, for mouse tracking within `WEClick`. This results in better CPU utilization, especially under Mac OS X.

• Fixed a bug whereby typing command-right arrow to go to the end of the second-to-last line of a document ending with a CR would incorrectly move the cursor down to the last line. Also fixed a similar problem involving command-left arrow. Thanks to Christopher Stone.

• Fixed a bug in the undo code that prevented `WEBeginAction`/`WEEndAction` from working correctly with `WECut` and `WEPaste`. Thanks to Chris Eplett.

• Incorporated fix for spurious strikethrough artifact in zero-width style runs such as blank lines. Thanks to Tom Bender.

• A new `WEFindPreviousObject` call complements the existing `WEFindNextObject` API:

```
pascal SInt32 WEFindPreviousObject
  (
```

```
    SInt32                inOffset,
    WEObjectReference *   outObject,   // can be NULL
    WEReference           inWE
  );
```

`WEFindPreviousObject` looks for the first embedded object *preceding* the specified offset: if one is found, a reference to it is returned in `outObject` and the byte offset to the embedded object in the text stream is returned as function result. If there is no object before the specified offset, `WEFindPreviousObject` returns `-1` as function result and sets `outObject` to `NULL`.

You can pass `NULL` in `outObject` if you're not interested in the object reference.

You can pass `-1` in `inOffset` as a shortcut for `WEGetTextLength ( inWE )`.

• A new field in the `WEPrintOptions` record, `firstPageOffset`, allows you to instruct WASTE to leave a blank area of specified height at the top of the first page. This effectively lets you specify different heights for the first page and for subsequent pages.

Also, a new `WEGetPageHeight` API lets you obtain the pixel height actually used by the printed text on any given page of a print session. This is usually equal to, or slightly less than, the page height, except possibly for the last page. When called for the last page in the range, this call allows you to position objects to be printed after the text.

```
pascal SInt32 WEGetPageHeight
  (
    SInt32                inPageIndex,
    WEPrintSession        inPrintSession
  );
```

The value returned by `WEGetPageHeight` for the first page includes the height of the blank band at the top specified using `firstPageOffset` (see above).

Thanks to Richard Aurbach for suggesting these enhancements.

• `WESetAttributes` and `WESetOneAttribute` now recognize additional meta-selectors that allow you to increase or decrease various paragraph-level attributes by a given delta throughout the specified range.

# New in WASTE 2.0b3 (October 2000)

• The fix for the `FetchFontInfo` problem in 2.0b2 was incomplete, causing the Carbon version of WASTELib to crash on OS 8.5 + CarbonLib 1.0.x. It turns out that CarbonLib 1.0.x does not export `FetchFontInfo` correctly in OS 8.5, even if that API is in fact available to classic apps (via the `FontManager` stub). Yes, that definitely looks like a CarbonLib bug. Now we only try to call `FetchFontInfo` if the system version is 8.6 or newer. Thanks again to the Macster team for reporting this problem.

• When styled Unicode import fails because the TEC can't recognize the target encoding hint, WASTE will now try again with the corresponding base encoding. This fixes a problem where WASTE running on a system prior to Mac OS 8.5 would fail to accept styled Unicode text created under Mac OS 8.5 or newer, because pre-Allegro TECs don't recognize the Euro variant of MacRoman.

• When converting traditional styled text to Unicode, WASTE will now silently discard partial source characters rather than stop the convertion process and return an error code. Partial (or incomplete) characters are errant halves of double-byte characters occurring at the end of a script run. They're uncommon, but may occur in traditional text

if the user forces a font change affecting the underlying text encoding.

• Mac OS X Public Beta seems to have a bug (Radar #2551463) in its implementation of `UpgradeScriptInfoToTextEncoding` (it will return `paramErr` with some perfectly legal combinations of input parameters) which creates several problems for WASTE. In particular, Japanese inline input was totally broken in OS X PB because of this. To work around this problem, this version of WASTE will use its own fallback implementation if `UpgradeScriptInfoToTextEncoding` fails, rather than propagating the error up the call chain.

• Fixed several issues with `U+FFFC OBJECT REPLACEMENT CHARACTER`s, used by WASTE as placeholders for embedded objects in Unicode text. WASTE now installs a custom fallback handler during Unicode to text conversion to remap `U+FFFC` to `ASCII 1`, which is historically used as a placeholder for embedded objects in traditional Mac encodings.

**Known problem**: Unfortunately, old (how old?) versions of the Text Encoding Converter treat `U+FFFC` as an undefined, rather than an unmappable, code point, stopping the conversion process without even giving the fallback handler a chance. To work around this problem, you should either make sure you're using a recent version of the Text Encoding Converter (1.5 is fine) or manually pre-filter the Unicode text replacing `U+FFFC` with `U+0001`.

• Changed `(x << 16)` to `BSL(x,16)` in a couple of spots in the source code to prevent problems when compiling with 16-bit ints. Thanks to Tom Bender.

# New in WASTE 2.0b2 (August 2000)

• The Carbon version of WASTELib 2.0b1 would crash when running on Mac OS 8.1 with CarbonLib 1.0.x, because it would try to call `FetchFontInfo` which is never available on OS 8.1 (even if the t-vector in CarbonLib is nonzero). Thanks to Chris Silverberg and Jason Toffaletti.

• Action sequences (the internal mechanism used to implement a sequence of undoable actions to be treated as a single action) were badly broken in that consecutive sequences were incorrectly merged into a single sequence. This affected both `WEBeginAction` / `WEEndAction` and undo of drag-and-drop actions. In this release, action sequences have been completely reimplemented using a different mechanism. Thanks to Dennis Ionov.

• `WEUndo` and `WERedo` will now return an explicit error code (`weProtocolErr`) if called within a `WEBeginAction` / `WEEndAction` pair.

• `WEEndAction` will return an error code (`paramErr`) if the `inActionKind` parameter is zero or negative.

• Automatic keyboard synchronization no longer occurs in inactive WASTE instances.

• Bottom borders now honor paragraph indents.

• `kCurrentSelection` can now be passed to `WEGetRunDirection`.

• Source code problem: `ItemReference` is now used instead of `DragItemRef` for compatibility with version 3.2 of the Universal Interfaces. Thanks to Mark Bernstein.

• Removed use of `TrackMouseLocation` to allow Carbon target to be compiled with version 3.3 of Universal Headers.

• Removed `WEPutCFString` (undocumented but exported by WASTELib 2.0b1). I will put it back after 2.0.

# New in WASTE 2.0b1 (July 2000)

• `WESetAttributes` (and `WESetOneAttribute`) would crash if passed out-of-range parameters in `inRangeStart` and `inRangeEnd`. Fixed. Thanks to Adam Woodworth for first reporting this.

• Fixed a problem with drag-and-drop text editing under Mac OS X where a WASTE instance would fail to detect drags originating from itself.

• Fixed a problem in `WEPut` where unstyled text inserted at a location other than the current insertion point was incorrectly picking up the null style. Thanks to Tom Bender.

• The Carbon version of WASTE was assuming `FetchFontInfo` is always available under Carbon, which is not the case for CarbonLib 1.0.x running under Mac OS 8.1. Thanks to Jason Toffaletti.

• The CFM termination routine in the WASTELib shared library (invoked automatically by the Code Fragment Manager when the connection to the library is closed) now performs some cleanup. Global data structures are released and Apple event handlers installed by WASTE are removed. In most cases, this cleanup is not necessary, as the connection to WASTELib is only closed when the client application quits, but it's a good idea for clients that manually load and unload WASTELib.

• Added a `WERemoveObjectHandler` API that complements the existing `WEInstallObjectHandler`.

• The `DisposeWE*UPP` routines now perform some stricter tests on their input and will do nothing if the opaque UPP you pass in is null or doesn't look like something that was created by the corresponding `NewWE*UPP` call.

• A couple of points in the source code (in `WEGetAttributes` and `WEMatchAttributes`) incorrectly assumed `sizeof(int) == 4`. Fixed, although this was only a problem if you compiled your own version of the library. Thanks to Tom Bender for spotting this.

• Fixed a problem where some WASTE calls were tapping temporary memory for potentially large blocks meant to be kept around after the completion of the call, even if you don't have the `weFUseTempMem` feature flag enabled. Specifically, WASTE would unconditionally tap temporary memory for embedded object data and for the undo/redo stacks. Now, if you have the `weFUseTempMem` feature flag disabled, all temporary memory allocated by WASTE calls will be released before the call returns. Thanks to Glenn Berntson.

• Upped the default memory partition allocated to the carbonated WASTE Demo to two megs (from 512K) to cope with increasingly memory-hungry builds of CarbonLib 1.1. This solves a number of silent failures reported with the previous version.

• The carbonated WASTE Demo now supports both the old, sessionless Carbon printing APIs and the new, session-oriented printing APIs introduced with CarbonLib 1.1, which are the recommended APIs for Mac OS X. The pre-built demo apps that comes with this distribution uses the old APIs for compatibility with CarbonLib 1.0.x.

• The WASTE Demo will now treat 'TEXT' files that begin with what looks like a byte-order mark (`0xFEFF` or `0xFFFE`) as Unicode text files.

• Fixed a bug in the WASTE Demo where it was calling `CheckMenuItem` with a zero item parameter.

• WASTE Demo: the 'Quit' menu item is now removed from the File menu when

running under Mac OS X.

- WASTE Demo: the icon for ‘utxt’ files sports a distinctive badge with the Unicode logo.
- WASTE Demo: added stationery pad support.
- WASTE Demo: added a format popup to the Save As dialog.

# New in WASTE 2.0a15 (April 2000)

- Fixed a couple of stupid, evil and probably crashing bugs. Spotlight is a godsend.
- The WASTE Demo will now save embedded objects in Unicode text files.

# New in WASTE 2.0a14 (April 2000)

• Added support for exchanging **styled Unicode text** with the external world. This completes my planned feature set for version 2.0 and makes this version a **beta candidate**.

Styled Unicode text in the WASTE 2.0 sense is a UTF-16 stream accompanied by the usual four WASTE 2.0 formatting scraps (`WEcf`, `WEst`, `WEpf` and `WEru`). A couple of considerations about these scraps. First, the text offsets in the `WEcf` and `WEpf` scraps are relative to the Unicode text, not to the original text, but they’re nonetheless **byte** offsets (and thus always even numbers). Secondly, the text encodings specified in the `WEst` scrap using the ‘ptxe’ tag are merely **hints** used to convert the Unicode text back to traditional Mac OS encodings, while the same tags specify the actual encoding of the corresponding text range when a `WEst` scrap is used for traditional text.

To obtain styled Unicode text for a given range, first use `WEStreamRange` to create the formatting scraps, then use the new API `WEGetTextRangeAsUnicode` to convert the specified range to Unicode while at the same time remapping offsets in the `WEcf` and `WEpf` scraps.

```
pascal OSErr WEGetTextRangeAsUnicode
  (
    SInt32               inRangeStart,
    SInt32               inRangeEnd,
    Handle               outUnicodeText,
    Handle               ioCharFormat,       // 'WEcf'
    Handle               ioParaFormat,       // 'WEpf'
    TextEncodingVariant  inUnicodeVariant,
    TextEncodingFormat   inTransformationFormat,
    OptionBits           inGetOptions,
    WEReference          inWE
  );
```

`WEGetTextRangeAsUnicode` is the low-level routine used internally by WASTE when you call `WEStreamRange` with the `kTypeUnicodeText` or `kTypeUTF8Text` selectors: `inRangeStart` and `inRangeEnd` specify a range to convert, `outUnicodeText` is a handle to hold the converted text, `ioCharFormat` and `ioParaFormat`, both optional, are handles to `WEcf` and `WEpf` scraps, respectively, to be remapped for the converted text. The `inTransformationFormat` parameter allows you to specify UTF-16 or UTF-8 Unicode. You can pass `weGetLittleEndian` in the `inGetOptions` parameter to specify that you want the text in little-endian byte-order (only applicable to UTF-16), and `weGetAddUnicodeBOM` to prepend a byte-order mark to the converted text.

`WEPut` now accepts styled Unicode text created in this way.

• WASTE 2.0 style tables (`WEst` scraps) can now optionally include embedded objects. Such style tables obviate the need for a separate `SOUP` scrap, and are the only supported way to save embedded object information for styled Unicode text.

To create a style table containing embedded object descriptions, call `WEStreamRange` with the `kTypeStyleScrap` selector, specifying `weStreamIncludeObjects` in the `inStreamOptions` parameter. `WEPut` will automatically recognize style tables containing embedded object information.

• Changed the WASTE Demo so that it can read and write (styled) Unicode text files.

**NOTE:** According to Apple's guidelines, such files have a 'utxt' file type and begin with a byte-order mark. Currently, only a few Macintosh applications can read such files. BBEdit Lite (as of version 4.6) and Style (as of version 1.6.1) are two of them. Microsoft Word 98 kind of works, but doesn't recognize Unicode paragraph separators (U+2029) correctly.

• Revised the **Intro to WASTE 2.0** document. A new section discusses compatibility issues and lists deprecated APIs.

• Converted all documentation files (including the one you're reading now) to Unicode format.

• Included a tentative fix for the long-standing "ghost trail" problem occasionally seen on Japanese systems, where a phantom glyph is still visible at the end of a line but the real character has been moved to the beginning of the following line by the line-breaking algorithm. Thanks to Koiso Norihito for suggesting this fix.

• Fixed a nasty bug in `WEStreamRange` (actually, in the previously undocumented `WEGetTextRangeAsUnicode`) that could corrupt memory and cause intermittent crashes.

• Slightly tweaked the Roman line-breaking algorithm in an attempt to make its behavior reflect as closely as possible what TextEdit does. Thanks to Dave Perman for suggesting this.

• Fixed a bug in `WEPut` introduced in version 2.0a13, where WASTE would fail to delete the specified text range if passed zero in the `inTextLength` parameter. Again, thanks to Koiso Norihito for catching and reporting this.

# New in WASTE 2.0a13 (March 2000)

• `WEPut` no longer honors the `weFReadOnly` feature flag.

However, other routines that call `WEPut` internally, like `WEPaste`, `WEInsert` and `WEInsertFormattedText` will return a `weReadOnlyErr` result code if called with the read-only feature on, as they did in all previous versions.

• `WESetAttributes` and `WESetOneAttribute` no longer honor the `weFReadOnly` feature flag.

• Added support for inserting Unicode text in the canonical UTF-16 format using `WEPut`. To insert Unicode text, pass `kTextEncodingUnicodeDefault` in the `inTextEncoding` parameter. Formatting flavors will be ignored. WASTE will use all of the installed scripts and fonts in an attempt to render Unicode characters not covered by the font used at the insertion point. For example, if you call `WEPut` to insert the Unicode character `U+0411` (`CYRILLIC CAPITAL LETTER BE`) and the font at the insertion point is Geneva but you have Osaka installed on your system, WASTE will use Osaka to render the Cyrillic character, because Osaka includes the corresponding glyph, while Geneva does not.

By default, `WEPut` assumes the Unicode text has a big-endian byte order and does not

begin with a byte-order mark (BOM). However, if you specify the `wePutDetectUnicodeBOM` in the `inPutOptions` parameter, WASTE will look for a byte-order mark (i.e., it will test the first two bytes against `0xFEFF` and `0xFFFE`), strip it off and endian-swap the remaining words if necessary. It is recommended that you always look for and add BOMs when exchanging UTF-16 text with external sources.

• In this version, the only allowed values for the `inTextEncoding` parameter in `WEPut` are `kTextEncodingMultiRun` and `kTextEncodingUnicodeDefault`. `WEPut` will fail with a `weInvalidTextEncodingErr` result code if any other value is passed. Future releases will accept additional encodings.

**NOTE**: the `inTextEncoding` parameter was simply ignored in 2.0a11 and 2.0a12.

• WASTE will now look for a 'utxt' flavor (Unicode text in UTF-16 format) in the clipboard and in incoming drags, but only if no 'TEXT' is found.

• Changed the WASTE Demo so it can read 'utxt' files.

• `WEBeginAction` and `WEEndAction` are now implemented.

• Fixed a bug introduced in version 2.0a10 that caused extra spaces introduced by intelligent cut and paste rules to be included in the selection when drag-moving a text range towards the beginning of the text.

• Fixed a printing bug that caused certain glyphs extending past the leftmost or rightmost edge of the printable area to get clipped. This would usually happen with italic text at large font sizes.

• Added declarations to the header files for the previously undocumented custom font mapper hooks. These are two callbacks, identified by the `weFontFamilyToNameHook` and `weFontNameToFamilyHook` selectors, invoked by WASTE when it creates and when it parses scraps that contain font names (the ‘WEst’ style scrap and the old ‘FISH’ font table). The prototypes for these hooks are the previously defined `WEFontIDToNameProcPtr` and `WEFontNameToIDProcPtr`.

• Added a special `kNullStyle` constant for `WEGetAttributes` (thanks to Kuniaki Nakamitsu for suggesting this). Passing this constant in the `inRangeStart` and `inRangeEnd` parameters allows you to retrieve the so-called “null” style, i.e., the style that WASTE would apply to the next typed character. When the selection is empty, the null style may not be the same as the style at the current insertion point. This happens because when the selection is empty, there is no proper range to which style changes can be applied, so WASTE needs an internal record to keep track of these changes. You can get the style at the current insertion point (as opposed to the null style) by passing `kCurrentSelection` in the `inRangeStart`/`inRangeEnd` parameters.

**NOTE:** This new behavior (changed from version 2.0a12) only affects `WEGetAttributes` and its simplified version `WEGetOneAttribute`. It does not affect `WEMatchAttributes`.

• In Carbon builds, WASTE will ignore result codes from the Toolbox routine `UpgradeScriptInfoToTextEncoding`, as a workaround for a bug in Mac OS X Developer Preview 3.

• Added a ‘carb’ resource to carbonated demo so it is launched as a Carbon app in OS X DP3.

• Fixed invalid memory access caught by Spotlight in `SLPixelToChar`.

• All calls to `NewRgn` now always check for a null result.

• Fixed old UPP macros in `WASTE.h` (thanks to Brian Stern). The old `NewWExProc` and `CallWExProc` macros are now mapped to `NewWExUPP` and `InvokeWExUPP`, respectively.

• Fixed a problem with UPPs in the Pascal interface file (thanks to Glenn Bertson for reporting this).

## New in WASTE 2.0a12 (January 2000)

• Fixed a nasty undo-related bug in `WEPut` (thanks to Koiso Norihito).

• Plugged a memory leak in `WEMatchAttributes`.

• Fixed a bug in `WEUseText`.

• Changed `WEMatchAttributes` and `WEGetAttribute` so that they use the "null style" when appropriate.

• `WEPut` no longer moves the selection range (leaving ghost hilites) when called for ranges that don't overlap the selection range.

• `WESetAttributes` and `WESetOneAttribute` could mess up the selection hilite when called for a range other than the selection range.

• Fixed a couple of problems in the Pascal interface file (missing `weTagStrikethrough`, misspelled `weTagRightIndent`).

• You can now pass `kCurrentSelection` to `WEGetRunInfo` and `WEGetParaInfo`.

• `WEGetAttributes` and `WEGetOneAttribute` now recognize two additional read-only tags: `weTagQDStyles` and `weTagTETextStyle`. The first tag returns the QuickDraw styles as a `Style` quantity; the second tag returns a `TextStyle` record.

• Added `WEGetObjectAtOffset` API.

• The Carbon version of WASTELib now supports command-clicking of URLs via Internet Config (even when linking against CarbonLib 1.0 at runtime).

• A static library for Carbon is now included.

## New in WASTE 2.0a11 (January 2000)

• WASTE 2.0a10 had some major problems when running under non-Roman system software. These problems are hopefully fixed in this version. Thanks to Koiso Norihito and Kuniaki Nakamitsu.

• The classic 68K static library in the a10 distribution was broken in many ways due to wrong compiler settings. This is fixed in a11. Again, thanks to Kuniaki Nakamitsu.

• Fixed a serious bug in `WENewPrintSession` that would corrupt the WASTE instance if its destination rectangle and the specified page rectangle had the same width.

• A new `WEPut` API lets you insert text accompanied by the new character formatting scraps introduced in WASTE 2.0a10, while providing a superset of the functionality exposed by `WEInsert` and `WEInsertFormattedText`.

```
pascal OSErr WEPut
  (
    SInt32              inRangeStart,
    SInt32              inRangeEnd,
    const void *        inTextPtr,
    SInt32              inTextLength,
    TextEncoding        inTextEncoding,
    OptionBits          inPutOptions,
```

```
    ItemCount             inFlavorCount,
    const FlavorType *    inFlavorTypes,
    const Handle *        inFlavorHandles,
    WEReference           inWE
  );
```

Some interesting properties of `WEPut`:

First of all, `WEPut` allows you to insert text at an arbitrary offset (not just at the current insertion point), optionally replacing the range specified by `inRangeStart` and `inRangeEnd`. Pass `kCurrentSelection` in these parameters to add the new text at the current insertion point, just like `WEInsert` would do.

The parameters `inFlavorCount`, `inFlavorTypes` and `inFlavorHandles` specify a set of formatting scraps to be applied to the inserted text. Several flavor combinations are possible. For example, you can specify a TextEdit-style "styl" scrap, optionally accompanied by a WASTE 1.x "FISH" font table. Or you can specify WASTE 2.0 formatting scraps ("WEcf", "WEst", "WEpf" and "WEru"). You can specify an optional "SOUP" scrap in both cases. Pass zero in `inFlavorCount` to insert raw text.

The `inTextEncoding` parameter specifies the encoding of the source text, and should be used when the encoding is not already implicitly specified by the formatting information, as is the case with WEcf/WEst pairs. If the formatting scraps implicitly specify the text encoding(s) used in the source text, these encodings override whatever you pass in `inTextEncoding`, so it is recommended that you pass `kTextEncodingMultiRun` in this situation. **NOTE**: the `inTextEncoding` parameter is ignored in this version of WASTE.

Passing `wePutIntCutAndPaste` in the `inPutOptions` parameter allows you to specify whether intelligent cut and paste rules should be applied to the inserted text or not. Unlike `WEInsert` and `WEInsertFormattedText`, `WEPut` ignores the state of the `weFIntCutAndPaste` feature flag.

Passing `wePutAddToTypingSequence` in the `inPutOptions` parameter allows you to insert text at the current insertion point without disrupting the ongoing typing sequence (if any). Normally, calling `WEPut` breaks any ongoing typing sequence and generates a new, separate undoable action. When you specify `wePutAddToTypingSequence` (and some suitable conditions are met), WASTE treats the new text somewhat as if it were inserted by repeated calls to `WEKey`. This option can be used to implement an auto-indent feature.

• Added support for the ~~strikethrough~~ style attribute. You can set and inspect the state of this attribute using the new `weTagStrikethrough` tag in conjuction with `WESetAttributes`, `WEGetAttributes`, etc.

• The WASTE Demo now saves extended formatting information in text files and reads it back using `WEPut`.

# New in WASTE 2.0a10 (November 1999)

• WASTE now keeps track internally of the **text encoding** associated with each style run. Like the script code, the text encoding can be derived from the font family number, but unlike the script code, the text encoding describes the mapping between code points and characters in a much more precise way.

• WASTE will now export character formatting information in a new scrap format, in

addition to the TextEdit-style ‘styl’ format. This scrap format mirrors the one used for paragraph-level formatting: it’s split into two interdependent flavors: a style table (flavor type ‘WEst’), which lists each unique style used in the text, and a character formatting array (flavor type ‘WEcf’) that maps text offsets to style table entries. The advantages of this new format are the following:

Memory-efficient. The size of the new flavors combined is typically less than the size of a ‘styl’ scrap.

Extensible. Styles are stored in a tree-like structure modeled after Apple event lists / records, and tagged with four-letter codes. When new styles are added to future versions of WASTE, older programs can still parse the tags they understand and skip the ones they don’t.

Preserves the new character formatting styles introduced in WASTE 2.0a9 (vertical shift and background color).

Stores font names (with their corresponding text encoding) rather than font family numbers, obviating the need for a separate font table (the ‘FISH’ flavor that can still be used to complement ‘styl’ scraps).

Accurately describes the text encoding of the accompanying text.

• A new read-only attribute selector, `weTagTextEncoding` (type `TextEncoding`), can be used to retrieve the text encoding associated with any character in the text. This attribute tag can is only valid in calls to `WEGetAttributes` or `WEGetOneAttribute`.

• A new `weTagTransferMode` attribute selector can be used to specify the QuickDraw text transfer mode to be used to combine text and background color. Most applications should stick with the default transfer mode traditionally used by WASTE, `srcOr`.

• Optimized some low-level routines to reduce the number of calls to `FetchFontInfo` (or `GetFontInfo`, in systems prior to 8.5) to the absolute minimum.

• Fixed a bug in the cursor-adjusting code introduced in version 2.0a8, which prevented the cursor from being turned into an arrow over selected embedded objects without a hover handler (or with a hover handler returning `weNotHandledErr`).

# New in WASTE 2.0a9 (October 1999)

• Introduced a new set of extensible, tag-based accessors for both character-level and paragraph-level attributes. These new accessors do not depend on rigid structures like `WETextStyle` and `WERuler`, so they allow for easier introduction of new attributes in future versions of WASTE. The functionality made available by the new attribute accessors is a superset of what you can do with `WESetStyle`, `WESetRuler`, `WEContinuousStyle` and `WEContinuousRuler`.

The new generic attribute-setter function is called `WESetAttributes` and it looks like this:

```
pascal OSErr WESetAttributes
  (
    SInt32               inRangeStart,
    SInt32               inRangeEnd,
    ItemCount            inAttributeCount,
    const WESelector     inAttributeSelectors [ ],
    const void * const   inAttributeValues [ ],
    const ByteCount      inAttributeValueSizes [ ],
    WEReference          inWE
```

```
  );
```

`WESetAttributes` can apply any number of attributes at once (even a mixture of character-level and paragraph-level attributes) on any text range (not just the selected text) as a single undoable action. You pass it three arrays, each one having the number of items specified in `inAttributeCount`: `inAttributeSelectors` is an array of four-letter tags specifying the attributes to be applied, `inAttributeValues` is an array of pointers to the corresponding attribute values, and `inAttributeValueSizes` specifies the size in bytes of each attribute value.

A conveniently simplified version of this call, `WESetOneAttribute`, can be used when there's only one attribute to set:

```
pascal OSErr WESetOneAttribute
  (
    SInt32              inRangeStart,
    SInt32              inRangeEnd,
    WESelector          inAttributeSelector,
    const void *        inAttributeValue,
    ByteCount           inAttributeValueSize,
    WEReference         inWE
  );
```

Unsurprisingly, the new generic attribute-getter function is called `WEGetAttributes` and is declared like this:

```
pascal OSErr WEGetAttributes
  (
    SInt32              inOffset,
    ItemCount           inAttributeCount,
    const WESelector    inAttributeSelectors [ ],
    void * const        outAttributeValues [ ],
    const ByteCount     inAttributeValueSizes [ ],
    WEReference         inWE
  );
```

This call lets you retrieve any number of attributes at once for any given character in the text. This call, too, has a conveniently simplified version to get just one attribute, `WEGetOneAttribute`.

You pass a single offset to `WEGetAttributes`, rather than a range, because the same attribute can have multiple values over an arbitrary range. In fact, applications often need to determine whether a given attribute value, or a set of attribute values, are present over a range, and whether they're continuous. The old `WEContinuousStyle` and `WEContinuousRuler` APIs addressed this need, but had various shortcomings. The new `WEMatchAttributes` is an attempt to overcome those shortcomings and is declared like this:

```
pascal OSErr WEMatchAttributes
  (
    SInt32              inRangeStart,
    SInt32              inRangeEnd,
    WESelector          inAttributeSelector,
    ByteCount           inAttributeValueSize,
    ItemCount           inArraySize,
    const void *        inValueArray,
    Boolean             outWhichValuesArePresent [ ],
    Boolean *           outIsContinuous,
```

```
    WEReference          inWE
);
```

You pass it an array of homogeneous attribute values (i.e., an array of fonts, or an array of colors) to be matched against existing attribute values over a given selection range. `WEMatchAttributes` will fill in an array of Booleans, setting the entries corresponding to attribute values which are represented in the specified range. It will also set `outIsContinuous` to true if the specified attribute has only one continuous value across the whole range. Applications can use `WEMatchAttributes` to mark Font, Size and Style menus according to Apple's Human Interface Guidelines, which dictate the use of checkmarks for continuous styles, and dashes for discontinuous styles.

Here are the character-level attribute selectors (four-letter tags) currently supported by the new accessors, along with the corresponding data types:

`weTagFontFamily` (type `FMFontFamily`, or `SInt16` for pre-Sonata headers) specifies the font family.

`weTagFontSize` (type `Fixed`) specifies the font size in PostScript points as a fixed-point (16:16) quantity for future extension, even if the fractional part is currently ignored by WASTE.

`weTagBold`, `weTagItalic`, `weTagUnderline`, `weTagOutline`, `weTagShadow`, `weTagCondensed` and `weTagExtended` (type `Boolean`) specify the corresponding QuickDraw styles. Since these are considered separate attributes, you can selectively turn one on while turning another off at the same time and leave every other style alone, all with a single call to `WESetAttributes`. Okay, you could do the same thing using `WESetStyle` with the `weDoFaceMask` selector, but the new scheme is more straightforward. Note that `weTagCondensed` and `weTagExtended` are mutually exclusive (setting either one automatically clears the other).

`weTagPlain` (type `Boolean`) is really a meta-attribute specifying the absence of all QuickDraw styles. Setting this attribute to `true` removes all QuickDraw styles, while setting it to false has no effect. Matching this attribute against `true` with `WEMatchAttributes` will tell you whether any portion of the selection is plain, and whether the whole selection is plain. You couldn't determine this using `WEContinuousStyle`.

`weTagTextColor` (type `RGBColor`) specifies the foreground color applied to the text, while `weTagBackgroundColor` (type `RGBColor`) specifies the background color in effect when the QuickDraw text-drawing primitive is invoked. **Warning**: the background color is an experimental feature that may be removed in future versions, so don't rely on it. The selection hilite doesn't display well on text with background color, and there is currently no way to remove the background color once it's set.

`weTagVerticalShift` (type `Fixed`) specifies a vertical dislocation from the baseline applied to the affected glyphs or embedded objects, expressed in PostScript points as a fixed-point quantity (the fractional part is currently ignored). Positive values move the affected glyphs up, while negative values move them down. This attribute can be used to implement subscripts and superscripts.

And now the paragraph-level selectors:

`weTagAlignment` (type `WESelector`) specifies the paragraph alignment. Its value can be one of `weTagAlignmentDefault` (align according to system direction), `weTagAlignmentLeft`, `weTagAlignmentCenter`, `weTagAlignmentRight` or `weTagAlignmentFull` (full justification).

`weTagLeftIndent` and `weTagRightIndent` (type `Fixed`) specify the amount of extra horizontal space between the paragraph and the left and right margins of the destination rectangle. `weTagFirstLineIndent` specifies an extra margin for the first line only.

`weTagLineSpacing` (type `Fixed`) specifies the amount of vertical spacing between paragraph lines, relative to the line height: `0.0` represents the normal spacing, `0.5` stands for one-and-half spacing, and `1.0` signifies double line spacing.

`weTagSpaceBefore` and `weTagSpaceAfter` (type `Fixed`) specify the amount of extra vertical space before and after a paragraph.

`weTagDirection` (type `WESelector`) specifies the dominant (primary) line direction in a bidirectional script environment. Its value can be one of `weTagDirectionDefault` (arrange bidirectional text according to the system direction), `weTagDirectionLeftToRight` or `weTagDirectionRightToLeft`.

`weTagBottomBorderStyle` (type `WESelector`) can be used to add a horizontal border line below the last line of the paragraph. In order for this border to be drawn correctly, there must be some extra space after the paragraph, or more than single line spacing must be used. Currently supported border styles include thin, thick and dotted.

All the new attribute accessors accept `kCurrentSelection` as a meta-value to signify the current selection range.

• Updated the WASTE Demo to show the use of the new attribute accessors.

• Starting from this release, WASTELib contains three versions of the library merged into one file: the CFM-68K version, the “classic” PowerPC version, and the Carbon version. In order to use the Carbon version of the library, link your application against the “CarbonWASTELib” stub library.

• A new feature flag, `weFNoKeyboardSync`, allows you to disable the automatic font/keyboard synchronization normally performed by WASTE as per Human Interface Guidelines. This runtime flag replaces the old `WASTE_NO_SYNCH` compile-time switch.

• Carbon status: the carbonated WASTELib has been tested with the CarbonLib in beta and final candidate versions of Sonata (Mac OS 9.0) and seems as stable as the classic (non-Carbon) version.

# New in WASTE 2.0a8 (September 1999)

• Fixed a bug in `WEGetObjectSize` (thanks to Kuniaki Nakamitsu).

• Re-implemented some low-level plumbing for embedded objects. As a result, some calls, like `WEGetObjectOffset`, are now much faster than before.

• Added a new `WEGetObjectFrame` API that returns the bounding box of an embedded object, in long coordinates (as the object may be outside the Quickdraw space). It is illegal to call this new API from within a “new” handler.

• Added a new “hover” handler for embedded objects. This handler can be used to set the cursor when the mouse is over an object, or to display help balloons or status messages when the mouse enters and leaves an object. A hover handler looks like this:

```
pascal OSErr MyHoverHandler
  (
    SInt16              inMouseAction,
```

```
    Point                inMouseLoc,
    RgnHandle            inMouseRgn,
    WEObjectReference    inObjectRef
  );
```

The `inMouseAction` parameter can be one of `weMouseEnter`, `weMouseWithin` or `weMouseLeave`. The `inMouseLoc` parameter is the current mouse location, in local (port) coordinates. The `inMouseRgn` parameter is only valid when `inMouseAction` is `weMouseEnter` or `weMouseWithin`. Its initial value is the rectangular region (in local coordinates) enclosing the object. The handler can restrict this region to be a portion of the object frame: WASTE will call the handler again as soon as the mouse leaves this region. This is useful for complex objects that need to set the cursor to several different shapes.

The hover handler should set the cursor when `inMouseAction` is `weMouseEnter` (and optionally, when `inMouseAction` is `weMouseWithin`); it shouldn't touch the cursor when `inMouseAction` is `weMouseLeave`. To have WASTE take care of the cursor for you, return `weNotHandledErr` from the handler.

Hover handlers are invoked from `WEAdjustCursor`, so your application must call `WEAdjustCursor` and `WaitNextEvent` in the usual way for hover handlers to work correctly.

Here's a sample hover handler that turns the cursor into a hand when the mouse is over a picture:

```
pascal OSErr MyHoverHandler
  (
    SInt16               inMouseAction,
    Point                /*inMouseLoc*/,
    RgnHandle            /*inMouseRgn*/,
    WEObjectReference    /*inObjectRef*/
  )
{
    if ( inMouseAction == weMouseEnter )
    {
        SetCursor ( * GetCursor ( kHandCursorID ) ) ;
    }

    return noErr ;
}
```

Install the handler code like this:

```
static WEHoverObjectUPP upp = NULL ;

if ( upp == NULL )
{
    upp = NewWEHoverObjectUPP ( MyHoverHandler ) ;
}

WEInstallObjectHandler ( 'PICT', weHoverHandler, upp, myWE ) ;
```

• WASTE Demo: re-implemented live scrolling in Apple-blessed (and Appearance-dependent) way.

• WASTE Demo: implemented window transitions (aka zooming rectangles).

• Some routines, like `WEStreamRange` and `WEGetHiliteRgn`, now accept a `kCurrentSelection` (`-1`) meta-value, that can be used to signify the current selection range.

• Fixed a problem that caused `WEGetObjectOffset` to return `kInvalidOffset` (`-1`) in some rare circumstances.

# New in WASTE 2.0a7 (August 1999)

• Fixed a problem with the command-right arrow combination introduced in the previous version.

• Fixed a potentially crashing bug in `WEGetIndRunInfo` (thanks to Timothy Paustian).

• Added new printing APIs, described in a separate document.

• Added `WEFind`, a new string matching API. The implementation of this call is based on a variation of the Boyer-Moore string matching algorithm: the longer the search string, the faster the search. The implementation has not been tested with non-Roman script systems yet, but the final release will support them.

```
pascal OSErr WEFind
  (
    const char *      inKey,
    SInt32            inKeyLength,
    TextEncoding      inKeyEncoding,
    OptionBits        inMatchOptions,
    SInt32            inRangeStart,
    SInt32            inRangeEnd,
    SInt32 *          outMatchStart,
    SInt32 *          outMatchEnd,
    WEReference       inWE
  );
```

The search string is described by the first three parameters: `inKey` points to the first character of the string, `inKeyLength` specifies the length of the search string in bytes and `inKeyEncoding` tells WASTE how the search string is encoded (pass `0 = smRoman` if you're looking for a plain ASCII string). Currently, `inKeyEncoding` must be the script code of an installed and enabled script system. You can specify various search options with the `inMatchOptions` parameter, including `weFindWholeWords`, `weFindCaseInsensitive` and `weFindDiacriticalInsensitive`. The search can be limited to a given text range using the `inRangeStart` and `inRangeEnd` parameters.

On exit, `outMatchStart` and `outMatchEnd` specify the start and end offsets of the first occurrence of the search string, if one is found. If no match is found, `WEFind` returns a `weTextNotFoundErr` result code.

• Added `WERemoveUserInfo` to complement the existing `WESetUserInfo` and `WEGetUserInfo`.

• The WASTE Demo has been updated. Printing (both classic and Carbon) is now supported. Various window management routines have been rewritten according to modern Window Manager 2.0 guidelines. Text is now soft-wrapped (re-wrapped dynamically to the window width as the window is resized, à la SimpleText).

The WASTE Demo now requires Mac OS 8.6 or newer.

• Carbon status: to compile this release for Carbon you'll need version 3.3d2 or newer

of the unified Carbon/Sonata headers, which are not available to the general public at the time of this writing. I'm not sure I can report on other Carbon-related changes without breaking my NDA.

• **Where's the source code??** It's now distributed as a separate archive to registered users. If you're a freeware author and need a free copy of the latest source code, please drop me a note. If you're a registered commercial developer and haven't received your password yet, contact me ASAP.

# New in WASTE 2.0a6

• The work on WASTE carbonization continues, closely tracking the evolution of the Carbon specification and SDK. WASTE can now be compiled with the Carbon headers and linked against CarbonLib (or LiteCarbonLib).

• The WASTE Demo is back! The old demo application from the WASTE 1.3 Distribution has been updated for Mac OS 8.5 and Carbon. It also shows off some of the new features in WASTE 2.0.

• If your application links against WASTELib and you're planning to carbonize it, you may want to adopt some new APIs in 2.0a6 designed to help you create and dispose of UPPs for WASTE callbacks. These new APIs have names like `NewWEClickLoopUPP` and `DisposeWEClickLoopUPP`, and are meant to be used instead of `NewWEClickLoopProc` (and similarly named macros) and `DisposeRoutineDescriptor`. In a classic world, `NewWEClickLoopUPP` will return a pointer to a routine descriptor, just like the `NewWEClickLoopProc` macro. In a Carbon world, where routine descriptors are obsolete, `NewWEClickLoopUPP` returns the function pointer itself, although this may change in the future. Of course, if you treat UPPs as opaque data types, you no longer have to worry. These new APIs also obviate the need for special glue code for Pascal.

• Removed all dependecies on AEGizmos, since this library isn't likely to be supported under Carbon.

• The `weFInhibitICSupport` feature flag was broken and is now fixed (thanks to Linda Peting).

• A long standing bug involving the use of option-left-arrow with certain non-Roman scripts was hopefully fixed (thanks to James Jennings).

# New in WASTE 2.0a5

• WASTELib is now fat (PPC/CFM68K).

• Fixed a potential crasher in `WEContinuousStyle` inadvertently introduced in version 2.0a2 (thanks to Timothy Paustian).

• Added a new low-level API to retrieve style run information associated with a given style run index:

```
pascal void WEGetIndRunInfo
  (
    SInt32            inStyleRunIndex,
    WERunInfo *       outStyleRunInfo,
    WEReference       inWE
  );
```

This call is similar to `WEGetRunInfo`, but takes a style run index rather than an offset into the text. The valid range for the style run index parameter is `0` to `WECountRuns(inWE) - 1`.

• Added four new APIs for low-level access to paragraph run information. These APIs mimic existing APIs for style run information:

```
pascal SInt32 WECountParaRuns
  (
    WEReference       inWE
  );

pascal SInt32 WEOffsetToParaRun
  (
    SInt32            inOffset,
    WEReference       inWE
  );

pascal void WEGetParaRunRange
  (
    SInt32            inParaRunIndex,
    SInt32 *          outParaRunStart,
    SInt32 *          outParaRunEnd,
    WEReference       inWE
  );

pascal void WEGetIndParaInfo
  (
    SInt32            inParaRunIndex,
    WEParaInfo *      outParaInfo,
    WEReference       inWE
  );
```

A **paragraph run** is a sequence of one or more paragraphs sharing the same ruler (set of paragraph styles). `WECountParaRuns` returns the total number of paragraph runs in the text. `WEOffsetToParaRun` maps a byte offset into the text to a paragraph run index in the range `0` to `WECountParaRuns(inWE) - 1`. You call `WEGetParaRunRange` to determine where in the text a given paragraph run begins and ends (the same information is returned by `WEGetIndParaInfo`).

• The meaning of the `inDestinationKind` parameter passed to object streaming handlers has changed slightly. Previous versions of WASTE pass `weToSoup` in this parameter whenever they have to create a 'SOUP' scrap (a data type used to describe objects embedded in the text), regardless of whether the scrap is to be used internally (e.g., to support undo/redo) or whether the scrap is to be copied to a "public" location (i.e., the desk scrap or a drag). This makes it difficult for streaming handlers to determine whether the object data must be converted to a format suitable for public consumption or whether it can be left in a private format (of course, this issue doesn't affect simple handler sets, like PICT handlers, that use common formats in the first place and thus don't need customized streaming).

Starting from this release, WASTE will pass `weToScrap` (instead of `weToSoup`) to streaming handlers when it's building a SOUP scrap to be copied to the desk scrap, and `weToDrag` (instead of `weToSoup`) when it's building a scrap to be copied to a drag. On the other hand, `weToSoup` will still be used for scraps created internally for undo/redo purposes, and when the streaming handler is called from the now deprecated

`WECopyRange`.

Finally, when `WEStreamRange` is called to create a SOUP scrap, the low byte of the `inFlags` parameter is interpreted as a destination kind to be passed on to streaming handlers. This allows further customization of the SOUP creation process.

Thanks for Rainer Brockerhoff for suggesting these changes.

- Started work to make WASTE Carbon-compliant.