Developer Release Note: Text Encoding Converter Manager 1.5

(Sept. 17, 1999 - P. Edberg)

This note describes external changes from Text Encoding Converter Manager (TEC) version 1.4.3 (included with Mac OS 8.6) to TEC version 1.5 (included with Mac OS 9.0 a.k.a. Sonata).

A. General enhancements & fixes

1. Booting & initialization

a) Locating the Text Encodings folder.

If a U.S. localized version of TEC 1.4.3 (or earlier) was installed on a localized system that changed the name of the Text Encodings folder, TEC could not the locate the Text Encodings folder if loaded in early boot by the file system (i.e. on Mac OS 8.1 or 8.5.x booted from HFS+ volumes, or on Mac OS 8.6 and later regardless of boot volume format). TEC first tries FindFolder to locate the Text Encodings folder; if this fails, it tries a folder name obtained from one of its STR# resources. The version of FindFolder available in early boot on many CPUs is not fully functional and does not support the Text Encodings folder; the folder name in the STR# might also be wrong if the TEC localization does not match the system localization.

The early boot TEC functionality that the file system depends on does not require the Text Encodings folder, so all TEC needs to do is make sure that the folder is found by the time there are clients that do require it. In TEC 1.5, every time a new client makes a connection to one of the TEC libraries, the TEC initialization code checks whether the Text Encodings folder has been found; if not, it tries FindFolder as described above. By later in the boot process, FindFolder has become fully functional and can locate the Text Encodings folder.

b) Ensuring that HFS+ uses the correct TextEncodingVariant.

ResolveDefaultTextEncoding calls Gestalt to get the system version, which is needed in order to resolve meta-values for certain TextEncodingVariants. In early boot, Gestalt returns a system version of 0; this resulted in HFS+ using the wrong TextEncodingVariant in some cases. Now, if Gestalt returns a system version of 0, ResolveDefaultTextEncoding gets the system version directly from the 'vers' resource of the System file.

c) A non-purgeable resource needed only at initialization was not being released; fixed.

2. Loading and locking tables

The functions ConvertFromTextToUnicode, ConvertFromUnicodeToText, and ConvertFromUnicodeToTextRun do not move memory (unless, for one of the latter two, a client has installed a custom fallback handler that can move memory). The tables used by these functions—as well as the TextToUnicodeInfo, UnicodeToTextInfo, or UnicodeToTextRunInfo objects used by them—are relocatable objects. However, there are two cases in which the tables and converter objects may need to be locked:

- System components which handle volume formats that use Unicode (components such as HFS+ support, UDF support, and File Exchange) may need to call ConvertFromTextToUnicode or ConvertFromUnicodeToText at interrupt time. In this case, it is necessary to ensure that the necessary tables and converter objects can never move (since an interrupt might occur when they happen to be in the middle of moving).
- If a client has installed a custom fallback routine for ConvertFromUnicodeToText or ConvertFromUnicodeToTextRun and this routine can move memory.

a) Interrupt-safe calling (changes to private functions)

For the system components, versions of TEC before 1.5 provided a private function that was called to lock the necessary items in advance. However, it had some problems:

- It loaded and locked some tables that were not required for ConvertFromTextToUnicode or ConvertFromUnicodeToText
- It did not lock the converter objects. This could result in ConvertFromUnicodeToText locking them at interrupt time (bad!) for reasons described in (b) below.
- Tables that were supposed to be locked could still be unloaded if they were shared with other mappings.

TEC 1.5 improves the old private function as much as possible, but that function can’t address all of these problems, so TEC 1.5 also provides a pair of new functions that do fix all of these problems. The system components in Mac OS 9.0 use the new functions.

b) New option for custom fallback functions

In versions of TEC before 1.5, ConvertFromUnicodeToText and ConvertFromUnicodeToTextRun attempted to lock the tables and converter objects they used (if they were not already marked as locked) in case there was a custom fallback handler that moved memory. This was overkill; they did this for most tables even if no custom fallback handler was installed.

TEC 1.5 defines a new mask that can be used in the iControlFlags parameter of SetFallbackUnicodeToText or SetFallbackUnicodeToTextRun to indicate that the custom fallback routine does not move memory:

- `kUnicodeFallbackInterruptSafeMask = 1L << 2`

Now ConvertFromUnicodeToText and ConvertFromUnicodeToTextRun only lock each table and converter object they use if (1) it is not already locked; (2) there is a custom fallback routine installed; and (3) the kUnicodeFallbackInterruptSafeMask option was not specified when the fallback routine was installed. This saves a lot of unnecessary locking.

3. Unicode properties data

TEC needs certain types of information about each defined Unicode character. Some of this is standard property information specified by the Unicode consortium, and some is TEC-specific. TEC 1.5 improves the data format and provides external access to some of this information.

a) New UCGetCharProperty function

TEC 1.5 introduces the UCGetCharProperty function (in the TextCommon library) which provides access to several types of Unicode character data specified by the Unicode Consortium.

```
extern OSStatus UCGetCharProperty(const UniChar *charPtr,
                                  UniCharCount textLength,
                                  UCCharPropertyType propType,
                                  UCCharPropertyValue *propValue);
```

The Unicode character is passed as pointer and length in order to handle UTF-16 sequences (and potentially UCS-4 32-bit characters eventually as well). The specific type of information requested for the character is specified by the propType parameter, for which three public values are currently defined:

- `kUCCharPropTypeGenlCategory = 1`   // General category (enum value)

- `kUCCharPropTypeCombiningClass = 2` // Canonical combining class (0..255)
- `kUCCharPropTypeBidiCategory = 3` // Bidirectional category (enum value)

Enumerations are provided for the values that may be returned for General category and Bidirectional category.

b) Internal access to character properties

TEC 1.5 also converts the TEC-specific properties data it uses into the same improved format as used by UCGetCharProperty; it preloads all the property data it needs and uses an internal version of UCGetCharProperty for access to them. These changes speed up the ConvertFromUnicode functions by up to 35% (depending on text length, etc.). They also reduce the size of the UnicodeToTextInfo and UnicodeToTextRunInfo objects (which formerly included a properties cache which is no longer necessary).

4. Caching common converter objects

TEC 1.5 caches TextToUnicodeInfo and UnicodeToTextInfo objects for converting between kTextEncodingUnicodeDefault (default variant & format) and the Mac OS system encoding (i.e. the most common conversion). When CreateTextToUnicodeInfo or CreateUnicodeToTextInfo are asked to create objects for such a mapping, they will clone the cached object (and mark it as a clone so the Dispose and Change functions can treat it accordingly).

B. Other UnicodeConverter library enhancements & fixes

1. New option kUnicodeTextRunHeuristics

Add `kUnicodeTextRunHeuristics` option for ConvertFromUnicodeToTextRun. For now, what this does is: If `kUnicodeTextRunHeuristics` is set and `kUnicodeKeepSameEncoding` is not, we behave as if `kUnicodeKeepSameEncoding` is set until we encounter characters outside the range u0020-u003F, at which point we reset to the preferred encoding. This has the effect of not switching back to the preferred encoding for space, digits, and common punctuation that follow some text not in the preferred script (e.g., space between two Arabic characters).

2. Fixes

a) ConvertFromUnicodeToTextRun with KeepSameEncoding

In a sequence of ConvertFromUnicodeToTextRun calls using the same UnicodeToTextRunInfo with the `kUnicodeKeepInfo`, `kUnicodeTextRun`, and `kUnicodeKeepSameEncoding` options set, the current encoding state (for KeepSameEncoding) was not being preserved in the UnicodeToTextRunInfo from one call to the next. Fixed.

b) ConvertFromUnicode with long digit sequences

In the ConvertFromUnicode scanner, when checking for a possible fraction-slash sequence (digit + fraction-slash + digit), we now allow at most one digit before and after the fraction-slash (instead of an arbitrarily long digit sequence). This greatly speeds up conversion of long digit strings (by 100x or more).

C. Other TextCommon library enhancements & fixes

1. New function NearestMacTextEncodings

This function maps an arbitrary TextEncoding to the Mac OS encodings that have the most similar repertoire.

```
pascal OSStatus NearestMacTextEncodings(TextEncoding generalEncoding,
                                        TextEncoding *bestMacEncoding,
                                        TextEncoding *alternateMacEncoding);
```

- generalEncoding is an arbitrary non-Unicode encoding.
- The value returned in *bestMacEncoding is the Mac encoding with the closest repertoire match.
- The value returned in *alternateMacEncoding is a Mac encoding that may be more generally available but may not have as good a match (e.g. if bestMacEncoding is Mac VT100, alternateMacEncoding might be Mac Roman). The alternateMacEncoding parameter can be NULL if the caller does not need this information.
- A value of kTextEncodingUnknown (0xFFFF) is returned for *bestMacEncoding or *alternateMacEncoding if there is no appropriate encoding.

2. Other enhancements

a) Get encoding names in Unicode

GetTextEncodingName can now provide names in Unicode if requested (currently just kTextEncodingUnicodeV2_1 and kTextEncodingUnicodeDefault).

b) New TECInfo feature bits

New feature bits (`kTECAddTextRunHeuristics`, `kTECAddFallbackInterrupt`) are defined for the tecUnicodeConverterFeatures field of TECGetInfo to indicate the availability of the new UnicodeConverter options (`kUnicodeTextRunHeuristics`, `kUnicodeFallbackInterruptSafe`) described above.

3. Fixes

If UpgradeScriptInfoToTextEncoding was passed an optional iTextFontname parameter and the system default sorting order was not U.S. English (due to system localization or Text control panel setting), the check for font variants could fail. At best it could return a TextEncoding with an incorrect variant; at worst it could potentially hang. Fixed.

D. Other TextEncodingConverter library enhancements & fixes

1. New function TECSetBasicOptions

Added the TECSetBasicOptions function. This enables some of the options available in the UnicodeConverter to also be used by the TextEncodingConverter functions, by affecting the operation of the plugins (currently these options only affect the operation of the Unicode plugin).

```
pascal OSStatus TECSetBasicOptions(TECObjectRef encodingConverter,
                                   OptionBits controlFlags);
```

The only UnicodeConverter options currently supported for this are:

- `kUnicodeForceASCIIRangeMask`

- `kUnicodeNoHalfwidthCharsMask`

2. Fixes to handling of UTF-8

Before TEC 1.5, the Unicode plugin (which provides Unicode-handling functionality to the TextEncodingConverter functions) did not indicate that it could directly convert between UTF-8 and non-Unicode encodings via the UnicodeConverter, although the UnicodeConverter can in fact handle such conversions. As a result, conversions from, say, UTF-8 to Mac encodings typically created a path such as UTF-8 to UTF-16 (via algorithmic conversion) then UTF-16 to Mac encoding (via the UnicodeConverter). Not only was this inefficient, requiring the creation of intermediate buffers, but conversion errors in the middle of converting an intermediate buffer are not handled properly, e.g. they are not mapped back to the corresponding offset in the original source buffer, etc.

With TEC 1.5, the Unicode plugin now correctly specifies what it can handle directly, including conversions between UTF-8 and non-Unicode encodings and conversions to or from decomposed Unicode (`kUnicodeCanonicalDecompVariant`). Consequently, conversion from UTF-8 to Mac encodings, for example, is now a one-step conversion.

To make this work, the TextEncodingConverter handling of Unicode variants and formats was generalized, and some restrictions were imposed on the code in TECCreateConverter that searches for paths between the input and output encodings:

- A non-Unicode encoding is not allowed as an intermediate encoding between two Unicode encodings.
- An encoding with a non-zero format is not allowed as an intermediate encoding.

There are still some problems when converting to UTF-8 format for decomposed Unicode; these are covered by separate bugs.

3. Fixes to handling of Japanese encodings

a) Better input validation

Do better validation of input from EUC-JP, JIS X0208, and MacJapanese/Shift-JIS in the algorithmic converters. Fix the MacJapanese/Shift-JIS to JIS X0208 converter to correctly handle halfwidth characters xA0-xA9, and to handle controls (x00-x1F, x7F) by mapping them to space.

b) Handle MacJapanese non-standard characters

Change the algorithmic MacJapanese-to-EUC-JP and MacJapanese-to-JISX0208 converters to handle MacJapanese non-standard characters (i.e. not in Shift-JIS) by either remapping them to standard characters (for the one-byte additions) or replacing them with fallback characters (for the two-byte additions). Also made sure that the lower-priority output status `kTECOutputBufferFullStatus` is not returned from these converters (or the reverse ones) if there is a higher-priority input error that could be returned (i.e. `kTextMalformedInputErr`, `kTECPartialCharErr`).

4. Fixes/enhancements to handling of Chinese encodings

a) Fix HZ converter

Fix the algorithmic HZ to ISO-2022-CN converter to correctly handle the HZ ASCII-mode tilde-tilde and tilde-linefeed combinations, instead of reporting them as malformed input.

b) Add internet encoding name "CN-GB"

Add internet encoding name "CN-GB" as another alias for "EUC-CN" and "GB2312" in TECGetTextEncodingFromInternetName.

E. New encodings, mapping changes

1. Additional encodings supported

a) ISO 8859-3 (Latin-3)

b) ISO 8859-4 (Latin-4)

c) ISO 8859-15 (Latin-9)

d) Windows code page 1257 (BalticRim)

e) Windows code page 1258 (Vietnamese)

f) Limited/basic support for Mac OS Tibetan

2. CJK mapping changes

a) Strict mapping changes, MacJapanese standard variant

Change some strict mappings for MacJapanese standard variant to match SJIS/CP932

- Change mapping for x8150 from u203E to uFFE3; change mapping for xEB50 (vertical form) from u203E+uF87E to uFFE3+uF87E; map u203E loosely to x8150.
- Change mapping for x8163 from u22EF to u2026; change mapping for xEB63 (vertical form) from u22EF+uF87E to u2026+uF87E; map u22EF loosely to x8163.
- Consequently, change mapping for xFF from u2026 to u2026+uF87F.

b) Add missing mappings, MacJapanese PostScript variants

Add missing mappings for some vertical forms (xEE5F-xEE81) in MacJapanese PostScript variants (the corresponding glyphs are only in fancy PS fonts e.g. from Morisawa). Users previously could not input these via Unicode-based input methods like Kotoeri and ATOK.

c) Additional loose mappings from Unihan characters.

The Unihan database can be used to identify many groups of ideographic characters that are separately-encoded variant forms (e.g simplified vs traditional forms, etc.). When one character in such a group has a strict mapping to a particular traditional CJK standard (such as JIS X0208) but the others do not, the other Unihan characters in the group can unambiguously be loosely mapped to the same traditional-standard code point. This typically adds around 1700-2900 loose mappings from Unicode to the Mac OS CJK encodings. This implements a suggestion made by Ken Lunde at IUC-13.

3. New Euro sign variants for Mac Cyrillic & Ukrainian encodings

For Mac OS 9.0, a new Cyrillic Euro sign encoding replaces the previous versions of the Mac OS Cyrillic and Ukrainian encodings. This encoding is based on the former Mac OS Ukrainian encoding, but changes xFF from CURRENCY SIGN to EURO SIGN.

TEC accommodates this as follows :

a) Define three new TextEncodingVariant values for Mac Cyrillic.

- `kMacCyrillicCurrSignStdVariant = 1           // Old MacCyrillic`

(Russian & Bulgarian localized systems before Mac OS 9.0)
xA2 = CENT SIGN
xB6 = PARTIAL DIFFERENTIAL
xFF = CURRENCY SIGN

- `kMacCyrillicCurrSignUkrVariant = 2            // Old MacUkrainian`
  (Ukrainian localized systems and Cyrillic language kit before Mac OS 9.0)
  xA2 = CYRILLIC CAPITAL LETTER GHE WITH UPTURN
  xB6 = CYRILLIC SMALL LETTER GHE WITH UPTURN
  xFF = CURRENCY SIGN
- `kMacCyrillicEuroSignVariant = 3               // New MacCyrillic`
  (Slavic Cyrillic localized systems & Cyrillic language kit for Mac OS 9.0 and later)
  xA2 = CYRILLIC CAPITAL LETTER GHE WITH UPTURN
  xB6 = CYRILLIC SMALL LETTER GHE WITH UPTURN
  xFF = EURO SIGN

b) Treat MacCyrillic variant 0 (now called `kMacCyrillicDefaultVariant`) as a meta variant; on pre-9.0 systems, it resolves to `kMacCyrillicCurrSignStdVariant` (1), and on 9.0 and later, it resolves to `kMacCyrillicEuroSignVariant` (3).

c) Treat MacUkrainian as a meta value; on pre-9.0 systems, it resolves to MacCyrillic variant `kMacCyrillicCurrSignUkrVariant` (2), and on 9.0 and later, it resolves to MacCyrillic variant `kMacCyrillicEuroSignVariant` (3).

4. Unicode-to-Unicode mappings

Added tables that permit the UnicodeConverter to map from arbitrary Unicode (`kUnicodeNoSubset` variant) to maximally decomposed Unicode (`kUnicodeCanonicalDecompVariant`). To do this set up a UnicodeMapping with arbitrary Unicode as the unicodeEncoding and maximally decomposed Unicode as the otherEncoding, call CreateUnicodeToTextInfo, and then call ConvertFromUnicodeToText to perform the conversion. Note that a call to CreateTextToUnicode with this UnicodeMapping will fail.

F. Summary of interface-file changes

These changes are in Universal Interfaces 3.3 and later.

1. TextCommon.h

a) Add the following TextEncodingBase values

```
kTextEncodingISOLatin6 = 0x20A, // ISO 8859-10
kTextEncodingISOLatin7 = 0x20D, // ISO 8859-13, Baltic Rim
kTextEncodingISOLatin8 = 0x20E, // ISO 8859-14, Celtic
kTextEncodingISOLatin9 = 0x20F, // ISO 8859-15, 8859-1 + EURO etc
```

b) Add the following special TextEncodingBase value to indicate an unspecified or unknown TextEncoding (as in the NearestMacTextEncodings function).

```
kTextEncodingUnknown = 0xFFFF,  // Unknown or unspecified
```

c) Add the following TextEncodingVariant values for MacCyrillic variants.

```
kMacCyrillicDefaultVariant = 0,      // meta value
kMacCyrillicCurrSignStdVariant = 1,
kMacCyrillicCurrSignUkrVariant = 2,
kMacCyrillicEuroSignVariant = 3,
```

d) Add the following constants for feature bits in TECGetInfo.

```
kTECAddTextRunHeuristicsBit = 6,
kTECAddFallbackInterruptBit = 7

kTECAddTextRunHeuristicsMask = 1L << kTECAddTextRunHeuristicsBit,
kTECAddFallbackInterruptMask = 1L << kTECAddFallbackInterruptBit
```

e) Add the following typedefs and values for Unicode properties.

```
typedef SInt32 UCCharPropertyType;
enum {
     kUCCharPropTypeGenlCategory = 1,     // requests enum value
     kUCCharPropTypeCombiningClass = 2,   // requests number 0..255
     kUCCharPropTypeBidiCategory = 3      // requests enum value
};

typedef UInt32 UCCharPropertyValue;

// General Category enum values (for kUCCharPropTypeGenlCategory)
enum {
     kUCGenlCatOtherNotAssigned = 0, // Cn Other, Not Assigned
     ...
     kUCGenlCatSymbolOther = 31      // So Symbol, Other
};

// Bidirectional Category enum values (for kUCCharPropTypeBidiCategory)
enum {
     kUCBidiCatNotApplicable = 0,    // for now use this for unassigned
     ...
     kUCBidiCatOtherNeutral = 11     // ON Other Neutrals
};
```

f) Add the UCGetCharProperty function.

```
extern OSStatus UCGetCharProperty(const UniChar *charPtr,
                                  UniCharCount textLength,
                                  UCCharPropertyType propType,
                                  UCCharPropertyValue *propValue);
```

g) Add the NearestMacTextEncodings function.

```
pascal OSStatus NearestMacTextEncodings(TextEncoding generalEncoding,
                                        TextEncoding *bestMacEncoding,
                                        TextEncoding *alternateMacEncoding);
```

2. UnicodeConverter.h

a) New option for ConvertFromUnicodeToTextRun

```
  kUnicodeTextRunHeuristicsBit = 11
  kUnicodeTextRunHeuristicsMask = 1L << kUnicodeTextRunHeuristicsBit
```

b) New option for SetFallbackUnicodeToText or SetFallbackUnicodeToTextRun

```
  kUnicodeFallbackInterruptSafeMask = 1L << 2
```

3. TextEncodingConverter.h

Add the TECSetBasicOptions API.

```
  pascal OSStatus TECSetBasicOptions(TECObjectRef encodingConverter,
                                     OptionBits controlFlags);
```