Technical Note TE520
Script Manager Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic - questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&As can be found on the Macintosh Technical Q&As web site.

[Sep 01 1993]

Str2Format and "'" thousands separator

I'm having trouble with the Script Manager `Str2Format` function when the thousands separator in the Numbers control panel has been set to "'". Using "," or "." as the thousands separator works OK. What's going on?

You discovered an bug in the `Str2Format` function. This routine includes an integrity check of the number parts table. The integrity check returns the "bad number parts table" error if any two different entries (other than `tokLeftQuote` and `tokRightQuote`, and other than `tokLeadPlacer` and `tokLeader`; cf. ScriptEqu.a) coincide. As you might guess, the standard `tokLeftQuote` and `tokRightQuote` is the same ASCII 0x27 character as the "'" thousands separator. Don't use ' as a thousands separator--sorry!

Back to top

CharType class field depends on type

Date Written: 1/29/93

Last reviewed: 4/1/93

The `CharType` function seems to be returning an incorrect result (*Inside Macintosh* Volume V, page 307) when running under the Kanji system. When the `CharType` function is applied to the first character of a Kanji string, it tells us that it's a blank class character. Is there a bug in the `CharType` function?

___

Inside Macintosh Volume V documents only values for the Roman script. Chapters 3 and 5 of *The Worldwide Development: Guide to System Software* (available on the Developer CD, in the folder Technical Documentation: Localization Information) have more complete information about CharType and the field values it returns under various script systems.

The general rule is to first look at the type field. The interpretation of the class field depends on the type, and class values may be considered subtypes. For Kanji, a class value 3 does not mean `smPunktBlank` (as it does for the Roman script), but something else, depending on the value in the type field. If in your case you're getting type = 2 (meaning `KataKana`), the class would be something mysterious, related to the way input methods have to deal with the byte. As an application developer, you shouldn't have to look at it. Only if you get the type value `smCharPunct` = 0, should you check the class field for a value = 3.

Back to top

How to get absolute (GMT) time for document creation date

Date Written: 11/12/92

Last reviewed: 6/14/93

Note:
The information in this Q&A has been updated and incorporated into Q&A OPS21. Please see dlsDelta field in PRAM's time zone MachineLocation record.

There's actually a system-level call to find out where you are. It's a Script Manager call named `ReadLocation` (used by the Map control panel), which returns a structure giving you all the information you need. Here's a description of the call, copied from MPW 411:

```
pascal void ReadLocation(MachineLocation *loc)
  = {0x205F,0x203C,0x000C,0x00E4,0xA051};
```

In C:

```
pascal void ReadLocation(MachineLocation *loc);
```

These routines access the stored geographic location and time zone information for the Macintosh from parameter RAM. For example, the time zone information can be used to derive the absolute time (GMT) that a document or mail message was created. With this information, when the document is received across time zones, the creation date and time are correct. Otherwise, documents can appear to be created after they're read. (For example, someone could create a message in Tokyo on Tuesday and send it to Cupertino, where it's received and read on Monday.) Geographic information can also be used by applications that require it.

If the `MachineLocation` has never been set, it should be <0,0,0>. The top byte of the `gmtDelta` should be masked off and preserved when writing: it's reserved for future extension. The `gmtDelta` is in seconds east of GMT; for example, San Francisco is at minus 28,800 seconds (8 hours * 3600 seconds per hour). The latitude and longitude are in fractions of a great circle, giving them accuracy to within less than a foot, which should be sufficient for most purposes. For example, Fract values of 1.0 = 90deg., -1.0 = -90deg., and -2.0 = -180deg.. In C:

```
struct MachineLocation {
  Fract latitude;
  Fract longitude;
  union {
    char  dlsDelta;  /*signed byte; daylight savings delta*/
    long  gmtDelta;  /*must mask - see documentation*/
  }  gmtFlags;
```

The `gmtDelta` is really a 3-byte value, so you must take care to get and set it properly, as in the following C code examples:

```
long GetGmtDelta(MachineLocation myLocation)
{
  long  internalGMTDelta;
  internalGMTDelta = myLocation.gmtDelta & 0x00ffffff;
    if ( (internalGMTDelta >> 23) & 1 )  // need to sign extend
      internalGmtDelta = internalGmtDelta | 0xff000000;
    return(internalGmtDelta);
}

void SetGmtDelta(MachineLocation *myLocation, long myGmtDelta)
{
  char  tempSignedByte;
  tempSignedByte = myLocation->dlsDelta;
  myLocation->gmtDelta = myGmtDelta;
  myLocation->dlsDelta = tempSignedByte;
```

Back to top

System script and string-to-number conversion

Date Written: 9/11/92

Last reviewed: 6/14/93

Do NumToString and StringToNum work correctly regardless of the script chosen as the system script? When I attempt to use SANE to convert non-Roman digits from a dialog box editText item, SANE doesn't seem to like it.

SANE expects all digits to be in the range ASCII $30-$39, with $2D as a negative indicator. These ASCII values can be generated from any international script by using the Macintosh numeric keypad. The symbols 0 through 9 are internationally recognized as numeric values.

There are many additional ways to represent numbers on the Macintosh, including words (one, two, uno, dos), notations (dozen, hundred, million), ordinals (first, second, third), Roman numerals (I, II, III), symbols ([[pi]], e, i), and hexadecimal ($FF). Many languages have alternative numbering systems and special characters that represent numbers. In Symbol and double-byte fonts, there are special characters representing fractions (1/2, 1/4), superscripts, subscripts, numbers within circles, and so on.

While it would be nice to have routines that convert between ASCII numbers and alternatives such as longhand numbers (used when writing checks), Roman numerals (used for copyright year in movie credits), or local number systems (for formal documents), no such routines exist in the Macintosh Toolbox today. It would be possible but difficult for an application to custom-process numbers for each language and script. The Unicode Standard Reference, Volume 1, lists hundreds of different kinds of numbers -- and they're not all base 10.

Scripts that have alternative number character sets always support the universal single-byte ASCII digits as well. When a script has alternative numeric characters, the user generally types script-dependent numeric characters from the top row of the keyboard and the single-byte ASCII digits from the numeric keypad.

Although it doesn't translate the digits themselves, the Script Manager offers support for formatting a number into a local form. For example, Europeans often use a comma as a decimal point and a period as a thousands marker. Most countries have unique currency symbols. There are many different ways to represent numerical values for things such as date, time, and money. This kind of formatting information is in the international resources.

One way to do data validation is to use CharType and check for numeric characters. We can't guarantee that this has been implemented for all scripts, but it is correct for Roman and Japanese.

NumToString and StringToNum don't deal with international formats. Use the Script Manager routines Str2Format and Format2Str to get the text into a numerical form that SANE can deal with. See *Inside Macintosh* Volume VI, page 14-49, for details.

Back to top

Using FormatXToStr and FormatStrToX with Pascal switches

Date Written: 12/10/90

Last reviewed: 8/1/92

Why do the FormatXToStr and FormatStrToX Script Manager routines stop working when I use the Pascal -MC68881 switch?

Regular SANE extended numbers are 10 bytes long while MC68881 extended numbers are 12 bytes long, and the extra two bytes are right in the middle of every 68881 extended number. Appendix G "The SANE Library" in the Macintosh Programmer's Workshop (MPW) Object Pascal version 3.1 manual goes into detail about this. The FormatX2Str and FormatStr2X parse the extended number you pass them directly, and they can only parse 10-byte extended numbers. Fortunately, you can still use the -mc68881 option with these routines as long as you convert any extended numbers to 80-bit extended numbers before passing them to FormatX2Str and FormatStr2X. The SANE.p unit has routines to do this called X96toX80 and X80toX96 (incorrectly documented as X96to80 and X80to96 in the MPW Object Pascal manual). Because the extended80 and extended96 types aren't equivalent to the extended type as far as Object Pascal is concerned, you have to redeclare FormatX2Str and FormatStr2X to take these types. You can do this as follows:

```
FUNCTION FormatX2Str80 (x:             extended80;
                        myCanonical:   NumFormatString;
                        partsTable:    NumberParts;
                        VAR outString:  Str255):  FormatStatus;
    INLINE $2F3C,$8210,$FFE8,$A8B5;

FUNCTION FormatStr2X80 (source:      Str255;
                        myCanonical:  NumFormatString;
                        partsTable:  NumberParts;
                        VAR x:       extended80):  FormatStatus;
```

Call these routines instead of the originals. To call `FormatX2Str80`, all you have to do is this:

```
VAR
   x:             extended80; {96-bit extended number}
   myCanonical:   NumFormatString;
   partsTable:    NumberParts;
   outString:     Str255
```

Calling `FormatStr2X80` is just slightly more complicated because the extended number is passed by reference:

```
VAR
   x:           extended;   {96-bit extended number}
   x80:         extended80; {80-bit extended number}
   source:      Str255;
   myCanonical:  NumFormatString;
   partsTable:  NumberParts;

x80 := X96toX80 (x);
result := FormatStr2X80 (theString, realCanon, PartsTable, x80);
```

You should find that these calls now work properly with the -mc68881 option set. This of course means that you'll need two versions of the source code; one with the calls to convert between 96-bit and 80-bit extended numbers for use with the -mc68881 option and another one which just uses plain old 80-bit extended numbers for use when the -mc68881 option is turned off.

X-Ref: *Inside Macintosh* Volume VI, page 14-49.

Back to top

String2Date and Date2Secs conversion surprises

Date Written: 9/17/91

Last reviewed: 8/1/92

Do `String2Date` and `Date2Secs` treat all dates with the year 04 to 10 as 2004 to 2010 instead of 1904 to 1910?

—

Yes, the Script Manager treats two-digit years less than or equal to 10 as 20xx dates if the current year is between 1990 and 1999, inclusive. Basically, it just assumes that you're talking about 1-20 years in the future, rather than 80-100 years in the past. The same is true of two-digit 9x dates, when the current year is less than or equal to xx10. Thus, in 2003, the date returned when 3/7/94 is converted will be 1994, not 2094. This is all documented in Macintosh Worldwide Development: Guide to System Software, available from APDA (#M7047/A).

Back to top

FormatX2Str strings

Date Written: 11/6/91

Last reviewed: 8/1/92

Using the Script Manager to convert numbers to strings and vice versa, in any language, what's the best way to create the string to pass to FormatX2Str? Will strings using the characters: "#" or "0" or "." or "," work no matter what script is currently running, and if not, what can I do?

___

The number format string and canonical number format string mechanisms that you use with FormatX2Str and its kin is a strange design, for exactly the reason that you asked about. The number format string (the one with the characters such as "#" and "0") does not necessarily work right regardless of the current script. In fact, it doesn't even necessarily work right between localized versions within one script system. The canonical number format string does work between localized systems and between script systems. The strange thing is there's an easy way to store number format strings (usually in a 'STR ' resource), but no obvious way to store canonical number format strings. Here's what you can do when converting between numbers and strings:

When you convert a number format string to a canonical number format string with Str2Format on a U.S. system, it converts it from something like "###.###" to a canonical number format string that looks something like, "three digits, a decimal point, and three digits." On a German system, that same number format string would be converted to "three digits, a thousands separator, and three digits."

What you can do to get around this is to save the canonical number format string in a resource instead of the number format string. The canonical string stores things in a language- and script-independent way. Create this resource by writing a trivial utility program that takes a number format string and calls Str2Format to convert it into a canonical number format string, and then copy this into a handle and save it as a resource of a custom type, like 'NUMF'. In your real program, load the 'NUMF' resource, lock it, and then pass the dereferenced handle to FormatX2Str and FormatStr2X.

You can see this done in the ProcDoggie Process Manager sample from the 7.0 Golden Master CD. Take a look at the SetUpProcessInfoItems procedure in UProcessGuts.inc1.p file. You'll see that the 'NUMF' resource is loaded, locked, and then passed to FormatX2Str. The result is displayed in the Process Information window.

If your program is localized by nonprogrammers, then you might want to provide the utility that converts a number format string to a canonical number format string resource just in case they have to change the entire format of the string. Then they can install the new 'NUMF' (or whatever you choose) resource as part of the localization process.

Back to top

Code for truncating a multi-byte character string

Date Written: 1/24/92

Last reviewed: 8/1/92

I create a Macintosh file name from another file name. Since I am adding information to the name, I must make sure that it is within the 31 chars maximum allowed by the operating system. What I need is the equivalent of the TruncText command, except instead of dealing with pixel width, I want the width to be number of characters (31). I can trunc myself, but I'd rather do a proper "smTruncMiddle" and have it nicely internationalized.

___

If you're going to be adding a set number of bytes to the end of a existing string and you don't want the localized ellipsis (from the 'itl4' resource) between the truncated string and your bytes, then you can use this routine:

```
PROCEDURE TruncPString (VAR theString: Str255; maxLength: Integer);
{ This procedure truncates a Pascal string to be of length maxLength or }
{ shorter. It uses the Script Manager charByte function to make sure }
{ the string is not broken in the middle of a multi-byte character. }
  VAR
    charType: Integer;
  BEGIN
    IF Length(theString) > maxLength THEN
      BEGIN
        charType := CharByte(@theString[1], maxLength);
        WHILE ((charType < 0) OR (charType > 1)) AND (maxLength <> 0) DO
          BEGIN
            maxLength := maxLength - 1;
            charType := CharByte(@theString[1], maxLength);
          END;
        theString[0] := chr(maxLength);
      END;
```

If you want the localized ellipsis (from the `'itl4'` resource) between the truncated string and your bytes, or you want the localized ellipsis in the middle of the combined strings truncated to a specific length, then you can use this routine:

```
FUNCTION TruncPString (maxLength: Integer; VAR theString: Str255;
truncWhere: TruncCode): Integer;
{ This function truncates a Pascal String to be of length maxLength or }
{ shorter. It uses the Script Manager TruncString function which adds }
{ the correct tokenEllipsis to the middle or end of the string. See }
{ Inside Macintosh Volume VI, pages 14-59 and14-60 for more info. }
  VAR
    found: Boolean;
    first, midPoint, last: Integer;
    tempString: Str255;
    whatHappened: Integer;
  BEGIN
    found := FALSE;
    first := 0;
    last := TextWidth(@theString[1], 0, Length(theString));
    IF Length(theString) > maxLength THEN
      BEGIN
        WHILE (first <= last) AND NOT found DO
          BEGIN
            tempString := theString; { tempString gets destroyed every }
                                     { time through }
            midPoint := (first + last) DIV 2;
            whatHappened := TruncString(midPoint, tempString, truncWhere);
            IF whatHappened < smNotTruncated THEN
              BEGIN { ERROR, bail out now }
                TruncPString := whatHappened; { return error }
                Exit(TruncPString);
              END
            ELSE IF Length(tempString) = maxLength THEN
              found := TRUE
            ELSE IF Length(tempString) > maxLength THEN
              last := midPoint - 1
            ELSE
              first := midPoint + 1;
          END;
        theString := tempString;
        TruncPString := whatHappened; { will always be smTruncated }
                                      { in this case }
      END
    ELSE
      TruncPString := smNotTruncated; { the string wasn't too long }
```

Back to top

Character type and subtype values within the Kanji system

Date Written: 11/17/89

Last reviewed: 8/1/92

What are the values of character type and subtype with the Macintosh Kanji system?

___

For Roman, these are the values of character type:

```
Punctuation         0
ASCII               1
```

For KanjiTalk, the values are the same as Roman, with the addition of:

```
Katakana            2
Hiragana            3
Kanji               4
Greek               5
```

In Roman, the subtype field is interpreted as:

```
Normal punctuation  0
Numeric             1
Symbols             2
```

The KanjiTalk subtype values are the same as Roman except if the character type is Kanji, in which case the subtype field takes these values:

```
JIS Level 1         0
JIS Level 2         1
```

Finally, for KanjiTalk, the character direction field is replaced by the In-ROM field. It is 1 if the character is in the ROM card and 0 otherwise.

Back to top

Interfacing a Macintosh application with Map CDEV data

Date Written: 8/17/92

Last reviewed: 10/11/92

How can I provide my users with a "hook" to access the geographical database in Apple's "Map" Control Panel from my application?

___

There's no supported way of accessing the geographical database contained in the "Map" Control Panel. Here are some hints, however (just to satisfy your curiosity):

The data are stored in a resource of type `'CTY#'`, ID=-4064, in the Map cdev. The resource format is a list of word aligned (variable length) city entries, preceded by an integer indicating the number of entries. Each entry has the format

- [`Integer`] length in bytes of the entry
- [`Longint`] latitude in Fract; north = +
- [`Longint`] longitude in Fract; east = +
- [`Longint`] GMT difference in seconds; east = +
- [`Longint`] (reserved; set to 0)
- [`PascalString`] name of the city.

Back to top

Downloadables

Acrobat version of this Note (K). Download

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code