Technical Note TE510
International Resource Q&As

CONTENTS

Downloadables

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&A's can be found on the Macintosh Technical Q&A's web site.

[Sep 01 1993]

Determining the language being used to enter text

Is there a way to tell what language is being used on the Macintosh? I know how to find the script and other international items, but the language being spoken would be a very useful thing to know.

—

It depends on what you mean by "language." It's impossible to determine what language the user is typing in without doing a high-level analysis of what is being typed, as you probably know.

One way that's been suggested in the past is to check on the current KCHR by calling GetScript with an smScriptKeys verb. This returns the resource ID of the currently active KCHR. All international systems come with a U.S. KCHR and possibly others, such as the Romaji and Kana KCHRs in the Japanese systems and the alternative Roman KCHRs in the various German and French systems. The thought was that the user will most likely choose the U.S. KCHR to type in English, and choose the French KCHR to type in French. One problem with this is that it isn't necessarily true. You can type English with the French KCHR and you can type French with the U.S. KCHR, and many people do. A much larger problem is that the KCHR IDs aren't standardized within a script system range. Because Apple defines a range of resource IDs for each script system for the international resources, you can tell what script system a KCHR is for, and you can even match up the KCHR ID with the standard KCHR IDs that Apple defines, but there's nothing wrong with someone creating their own KCHR and giving it any ID they want, as long as it's in the proper range for the script system they've written it for. If that's the currently active KCHR, its resource ID tells you what script system it's for, but it won't tell you anything about what language it was intended for. For this reason, this method is frowned upon now.

In fact, there really is no reliable way of knowing the language that's being used. The best you can do currently is to find out what the system is localized for. This is done by grabbing the vers resource ID 1 from the System file. In this resource is what was called the country code (the term "country code" is obsolete, in favor of "region code") which indicates what region the system is localized for. These region codes are defined in Packages.p in MPW, and have names like verUS and verFrance and the like.

There's another way that you might want to consider. One of the GetScript verbs is smScriptLang. This returns the language code of the specified script system for the current system. If you pass smRoman as the script code to GetScript, it'll return langFrench on a French system, langGerman on a German system, langEnglish on a U.S. or U.K. or Australian system, and so forth. If you pass smJapanese as the script code to GetScript, it'll return langJapanese. Interestingly, if you pass smRoman as the script code to GetScript when a Japanese script system is running, it'll return langEnglish. All non-Roman script systems return langEnglish if you pass smRoman as the script code to GetScript. The language constants are in Language.p in MPW. You'll probably what to combine this GetScript call with a call to GetEnvirons to find out what the currently active script is. It might look something like this:

```
(* Get the currently-active keyboard script *)
keyboardScript := GetEnvirons (smKeyScript);

(* Now get the language that the keyboard script corresponds to *)
```

In summary, there's no system support for retrieving the language that the user is typing, nor is there any reliable support for retrieving the language related to any KCHR. But, you can find the region code of the active system, and you can find the language associated with the active script. Hopefully, that's enough information to be useful to you.

Back to top

RelString & EqualString vs International Utilities

Date Written: 1/25/91

Last reviewed: 8/1/92

What is the difference between RelString and EqualString? What should Macintosh developers use when sorting? Do you suggest having an option for the international sort?

___

RelString and EqualString are mainly intended for the File Manager. The File Manager uses them for quick-and-dirty string comparison so that it knows how to return files ordered alphabetically when you use indexed File Manager routines, and so that it can detect file name collisions. Beyond that, RelString and EqualString aren't localizable or extensible.

The International Utilities string comparison routines are localizable and extensible. They use information in the active 'itl2' resource to determine how the characters are sorted. Most localized systems come with their own 'itl2' resource, so string comparisons are done correctly for the region for which the system is localized. Because RelString and EqualString stay the same for all these regions, you'll probably find some cases in which strings are compared incorrectly by these routines.

One important place where RelString and EqualString don't work very well is with the new characters in the extended Macintosh character set. When the LaserWriter was introduced, the LaserWriter fonts used the extended Macintosh character set, which added many new characters, including several new uppercase characters with diacriticals. In system software version 6.0.4, the International Utilities were updated to take advantage of these new characters. For example, the uppercase "E" with a grave accent first appeared in the extended Macintosh character set. With 6.0.4, the lowercase and uppercase "E" with a grave accent were considered to be equal in primary ordering, and unequal in secondary ordering, which is correct. Even today, RelString and EqualString think in the old Macintosh character set, erroneously believing that lowercase and uppercase "E" with a grave accent have nothing to do with each other.

Back to top

Macintosh PRAM's MachineLocation dlsDelta field

Date Written: 1/25/91

Last reviewed: 8/1/92

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.

Back to top

Macintosh PRAM's MachineLocation dlsDelta field

Date Written: 1/25/91

Last reviewed: 8/1/92

How is the dlsDelta field in PRAM's time zone MachineLocation record used and set?

___

Currently, the dlsDelta field is not being used by Macintosh system software, nor is its meaning defined. There are plans to use it in the future, so it's important that you preserve its current value if you ever use WriteLocation to set the value of gmtDelta. See the description of the WriteLocation routine in "WorldWide Development: Guide to System Software," available on the latest developer CD and from APDA (#M7047/A), for details on getting and setting gmtDelta while leaving dlsDelta intact. In short, the code looks like this:

```
VAR
    myLocation:     Location;
    myGMTDelta:     LongInt;
    tempSignedByte: SignedByte;
:
tempSignedByte := myLocation.dlsDelta;
myLocation.gmtDelta := myGMTDelta;
```

Back to top

Why 1904 is Macintosh Time base

Date Written: 9/6/91

Last reviewed: 8/1/92

The global variable Time contains the number of seconds since midnight, Jan. 1, 1904. Why was the year 1904 chosen ?

___

The ability to go back in time is one consideration. You would not want to start the clock from, say, 1984. You also want to go ahead in time a good amount, of course. So, the clock start date needs to put our current date somewhere in the middle of the clock's range.

So what is the clock's range? Since the clock chip has a four byte counter which is incremented each second, they had 4,294,967,295 seconds to work with, or approximately 136 years. This would make the Macintosh clock run out in 1904+136 = 2040. The maximum value, $FFFFFFFF, corresponds to 6:28:15 a.m., February 6, 2040.

Given the possible range of years/leap years/nonleap years (every fourth year, but not if at the end of a century, except at the end of every fourth century, which is a leap year) and the date when the clock will run out (2040) - the "leap year code" in the Macintosh only has to deal with the rule "every fourth year is a leap year" because none of the possible Macintosh dates violate that rule! Remember, 2000 is evenly divisible by 400, so it IS a leap year. 1900 is not. If they started at 1900, they would have to use a different algorithm that accounts for "non-leap year" leap years. Some other clocks start on 1901.

Why did they start on 12:00:00, January 1, 1904? Well, it probably has to do with 1904 being the first leap year after a "non-leap year" leap year (1900). So, the year was chosen for mechanical (4 byte limit on number of seconds) and pragmatic (you only want to use one algorithm to figure out the date et al) considerations.

So, back to the future...

X-Ref:

Chapter 4, "Worldwide Guide to System Software"

Back to top

System 7 and 'itl1' resources

Date Written: 9/16/91

Last reviewed: 8/1/92

System 7 doesn't recognize some of my 'itl1' resource changes, such as date abbreviation length, that are recognized by System 6.

___

The field that you refer to is not doing what you want because System 7 introduced an expanded 'itl1' resource, with several new fields, including arrays that contain the proper abbreviations of all the months and days. If you look in *Inside Macintosh* Volume VI on pages 14-87 thru 14-89 you'll find a description of this new resource as well as the rez

definition of it. Note the two new arrays, `abbrevDays` and `abbrevMonths`. If you were to modify these additional fields, you'd see the date in the finder windows change. (In fact, the abbrev length field does not seem to be used if these new fields are present.)

Back to top

'itl0' resource time1Suff...time8Suff field interpretation

Date Written: 11/4/91

Last reviewed: 6/14/93

How are the `'itl0'` resource `time1Suff`, `time2Suff`, ... to be interpreted? On the German system, these bytes seem to contain " Uhr Uhr." The correct 24-hour time suffix should be a single " Uhr." How do we know how many bytes of this 8-character entity are significant? Similar question for the mornStr and eveStr: are all 4 characters of each of these significant?

___

The `time1Suff`...`time8Suff` fields in the `'itl0'` resource of all localized systems divide those fields into two sections. The first four bytes (`time1Suff` through `time4Suff`) are used from 12:00 midnight through one second before noon, and the last four bytes (`time5Suff` through `time8Suff`) are used from 12:00 noon through one second before midnight. This is to take into account any system that has hours from 00:00:00 through 23:59:59, but has different time trailers for the morning and evening hours. I don't know of any that work this way offhand, though. The German standard is just to append "Uhr" onto the time, morning or evening, and it's separated from the time by one space. Any unused bytes just contain zero. For example, if the German time suffix was "Uh", then the time suffix fields would contain a space, "U", "h", zero, space, "U", "h", and zero. If there's no time suffix, then all eight bytes are zero.

The morning string and evening strings are done in a similar way. Each one holds a maximum of four bytes, and any unused bytes are filled with zeros.

By the way, you'll find that a lot of time suffixes are filled in with something even though the flags say that time suffixes aren't used. This is just localization garbage, which probably will be cleaned up someday.

Back to top

Downloadables

Acrobat version of this Note (K). Download
