Straw man proposal and flash.globalization – Differences

Mihai Niță | Globalization Architect

flash.globalization

flash.globalization

- It is something that is working
  - wrappers for all major APIs: Win32, Mac Core Foundation, POSIX, ICU, Java (thru JNI)
  - working today on many systems (Windows, Mac OS X, Linux, Solaris, WinMo, iOS, Android, Web OS, QNX)
- The API was reviewed in detail by our ActionScript “Public API Review Board” (PARB)
- We don’t claim it is perfect, or even a good model
- We learned some interesting lessons
- But please take a look at it and feel free to pick my brain

flash.globalization

- flash.globalization – Functionality
  - UTS #35 Locale identifiers (parse, map/match, extra info)
  - Number formatting
  - Currency formatting
  - Date & time formatting
  - Collation
  - Case conversion
- http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/globalization/package-detail.html

Our guiding principles – was really-really helpful to have them

- General good framework design principles
  - Make it easy to do the right thing
  - Discourage (but don’t prevent) the wrong thing
  - Be as consistent as possible
- Our own “rules of thumb”
  - Our developers are more important than us
  - But the end users are more important than our developers
  - If “feature X” is missing on Win and Mac, it is probably OK to be missing in our 1.0 version
  - Will try hard to give you “something”
    (and will tell you we did that)

Some differences

- Other stuff we did kind of differently
  - we force certain parameter for constructors especially for things that it is hard/impossible to get right (locale, some flags)
  - each formatter has a defaultLocale and a way to enumerate locales (like Java & ICU) => in some OSes it is possible to have different defaults for each kind of operation
  - for each formatter we have a requestedLocaleID and an actualLocaleID (a bit like ICU’s udat_getLocaleByType)
  - for generic locales (“fr”) we don’t throw exception (like .NET) and don’t use “¤” or silently use some base (like ICU) => we use the “best fit locale” and set actualLocaleID to it ("fr-FR")
  - error codes for each method call (if we do fallback)

Some differences

- Locale
  - no displayABC => most OSes don’t have this info
  - old UTS #35 (@ vs. -u-, co vs. collation) => we did the work before that
  - advanced mechanism to match locale lists (want vs. have)
- Collation
  - we use flags (ignoreCase, ignoreDiacritics, not strength) => easier to use
- TimeZone handling
  - We have nothing => poor OS support, no standard id, not strictly a formatting thing, DataTime object in Player would require some major changes

Some differences

- Date & time
  - no concept of “skeleton”, but we have predefined enums for long, medium, short forms (most APIs work this way)
- Number / currency
  - flags, not patterns and prefix/suffix=> tough to do on top of most OSes, and a bit “too much freedom.” They also interact with each other in “weird ways.”
  - No scientific support => little/no OS support, not quote possible in plain text (real locale-sensitive results look like this: $1.323,72 \times 10^{-23}$)
  - Currency and Number are handled by different classes
  - No percent support => little/no OS support, maybe in next version
- Message formatting
  - Support in the Flex ResourceManager (simple parameters, no plural, gender, etc.)
  - But there is a standard mechanism to store/load strings (also Flex ResourceManager)

Some ideas

Some ideas for ECMAScript

- Getting default locales:
  - User specified list of browser preferred locales (the one used in the Accept-Language field in the HTTP-Request header).
    This is the number 1 question we get in all presentations
  - Others? (OS UI language, browser UI language, OS settings for formatters)
- Matching locale lists (~RFC 4647, locale distance)
- String externalization?
- Character properties (isDigit, isLetter, isUpper, etc.)
- Normalization? (that info is there for IDN in many browsers)
- Take a look at the recent Microsoft contribution to jQuery

Adobe