
JavaOne℠
Sun's Worldwide Java Developer Conference

Internationali-zation

Asmus Freytag

Overview

- Background
- Requirements
- Approaches
  - Character Encoding (internal/external)
  - Locale Support
  - AWT: Input, Fonts, Localizing Applets
  - HotJava
- Delivery Stages

Background

- Users have easy access to documents worldwide, in any character set
- Servers can be accessed by users from anywhere, speaking any language
- Software can no longer be targeted to a single national market

~ The Internet pushes the envelope on Internationalization

User Requirements

- Display text data from any source
- Run localized apps/applets
  - Access localized web pages and applets by language

Programmer Requirements

- Create internationalized Apps/Applets
- Localize Apps/Applets easily

Char Data Type and Identifiers

- Java’s character data type is Unicode™
- Identifiers: any Unicode letter or digit
  - Currently spec'd as Unicode 1.1
  - Will be Unicode 2.0 as of JDK 1.1
- Remove limitations:
  - Current limit: 0000-00FF in PrintStream
- Deprecate:
  - Small # of APIs assume ‘byte[]’ as text

Character Encoding

- External data are not all in Unicode
- Class CharacterEncoding
  - Conversion functions for most common character sets
- Extensible
- Add: Character code conversion in
  - DataInputStream.readChar
  - DataOutputStream.writeChars

Locale

- Flexible locale model
- *NOT* global, but object-oriented
- Hierarchy of services rooted in Class LocaleDependent
- Allows definition of generic locale dependent services
- AWT Components carry locale designation

Fonts

- Currently only Latin 1 fonts.
- Abstract font names for native fonts:
  - For font name "Serif", Java runtime will try to use Serif type platform fonts for all Unicode glyphs.
  - If the glyph is not available, Java will display a substitute character.
- Future: provide combining fonts APIs

Input Method

- Java will support native Input Methods via native widgets used by awt.
- Future: access via Java specific rich text widgets, possibly API
- Future: support for platform indepen-dent input method desirable

User Interface Localization

- Original java UI approach wraps code and localizable data
  - Very flexible, but broken for localization.
- Short term: use property sheets
- Long term: JavaSoft is working with licensees to define a common, sharable serialization of classes, so GUI builders can support localization tools

HotJava

- HotJava strings and messages are kept in property files
- Currently displays HTML or documents encoded in ISO 8859-1
- HotJava 1.0 will be based on JDK 1.1 and leverage new i18n features to support display of multilingual text
- HotJava will support localized applets

Staged Release

- Java i8n features will be released in stages
- '1.1' :
  - CharacterSet support and Locale Model
  - Initial input and font support
- Future: UI localization, rich text, etc..

Summary

- Minimal subset immediatly
- Rich support later in stages
- Working with licensees to define and implement support
- Thank you. Any Questions?