



# Implementing a Globalization Library for Embedded Systems

Steve Cohen, EVP/CPO

20<sup>th</sup> International Unicode Conference  
January 2002

strategy • process • technology • results  
[www.basistech.com](http://www.basistech.com)



**BASIS**  
TECHNOLOGY

## Part I: Embedded System Introduction

20<sup>th</sup> International Unicode Conference  
Washington, DC January 2002

## Why Embedded Systems

- Ubiquitous computing
  - Smart devices
- Used for control of devices large and small
- Often have to be inexpensive and/or work in hostile environments

## Where are Embedded Systems

- Consumer devices (phones, watches, toys, home appliances, MP3 players)
- Office devices (printers, fax machines, communications equipment, PDAs, phones)
- Industrial devices (process control/PLC, monitoring, robotics)
- Military devices (weaponry, remote sensing, protective armor)
- Vehicles (engine control, safety, navigation, user features)

## What makes Embedded Systems difficult

- Physical Environment
  - Waterproof (watches), chemical proof (PLC), heatproof (thermometer)
  - Reliable (don't want a BSOD in a nuclear plant)
  - Power usage (PDA, GameBoy)
- Cost
  - Consumer devices are most cost sensitive
  - Commercial devices are, too
  - Computing is often NOT the reason for buying the device – this limits the amount that can be spent on the computer

## Computing Resource Constraints

- Hardware
  - RAM (heap and stack)
  - ROM
  - Disk
  - CPU speed
  - CPU cycles (real-time systems)
- Tools
  - Specialized Operating Systems
  - Limited toolset (Assembly, C, EC++)
  - Difficult to debug (often don't have console, much less graphical debugger)
  - Simulators (hardware and/or software)
- Generally a few generations behind general purpose computing resources

## Embedded Operating Systems

- WindRiver VxWorks
- Lynx LynxOS
- RTLinux
- Microware OS-9000
- Palm OS
- Microsoft WinCE



**BASIS**  
TECHNOLOGY

## Part II: Unicode in Embedded Systems

20<sup>th</sup> International Unicode Conference  
Washington, DC January 2002

## What was Unicode designed for

- General purpose computing
- Comprehensive coverage of all the world's scripts
- Design principles do not mention space (though efficiency is discussed)
- Specify rules to ensure consistent data processing across applications (collation, compression, ...)
- Wide character storage (more RAM use)

## The result: Unicode is big

- How big? Full Rosette, with 150 encodings (don't forget **many** variants) requires 11 MB of storage on disk. Of this 10 MB is just the data
- Most of this data is for transcoding

## **Part III: Embedding Rosette C++ Library for Unicode**

- Client need: Unicode support in an office automation device
  - Primarily for display screen use, to display names and addresses on LCD screen

## Introduction to Rosette

- Cross platform Unicode library
- Uses C++ features: templates, exceptions
- Original design target for desktop and client-server applications
- Design tradeoffs generally favor speed over space
- Besides, the data is simply BIG

## Embedded Rosette Example

- Office automation equipment
- LynxOS on MIPS CPU
- gcc, but no exceptions
- could be worse: Embedded C++ specifies no templates, exceptions, MI, RTTI, namespaces, STL

## Requirements

- Static (disk or ROM) space unknown, but “as small as possible”
- Stack usage <2 KB
  
- Features:
  - Japanese and 6 European language transcodings
  - Collation
  - Character properties
  - Normalization

## First step: Measure where we start

- Disk size: 11 MB
- Stack usage: 20 KB
- Heap Usage: Variable, depending on application use

## Reducing static size

- Remove unneeded encodings
  - Conditionally compile out most of over 150 encodings that Rosette includes
- Make features configurable
  - #ifdefs galore
- Eliminate ‘template bloat’
  - gcc flags: -frepo, -fno-implicit-templates
- Reduce internal data structure size
  - gcc flag: -fsmall-enums

## Reducing stack usage

- Allocate space on the heap instead
- Defined a buffer template class to encapsulate the buffers
  - Permits a `#ifdef` to change behavior from stack to heap

```
char a[2048];
```

vs.

```
char* a = new char[2048];  
delete [] a;
```

## Result: A slimmer Rosette

- Static size: 640 KB peak usage
  - Code and data
  - SJIS transcoding table is 200Kb of this
- Stack usage: 650 bytes!
- Client's requirements met, on an 8 week schedule!

## **Result: High reliability**

- Rosette unit test coverage >90%
- Same code base is used for embedded version
- Nightly build and testing, on 14 platforms

## Can we make it even smaller?

- Smallest: reduce 200K SJIS table to 85K using compression
- #ifdef more features if not required by application



**BASIS**  
TECHNOLOGY

## Part IV: More Unicode in embedded systems?

## Consider the devices

- Portable devices: Email, SMS, text display (e-books), MP3 players playlist
- Office automation: Email, web access, Voice, user controls
- Retail: POS terminals
- Appliances (kanji-capable internet toaster?)

## Phones: One example

- Today, mobile phone makers would like to have a single platform to ship to different locales
- Tomorrow (literally), mobile phone users will send multilingual data to phones in other locales
- Next week (literally), mobile phone users will use the same phone in several locales (eg. GSM adoption by Verizon)

## PDA's: Another example

- Synchronize data with enterprise and web software
- Connect via wireless to the same data sources (like mobile phones)
- Capable of running “office applications” (Pocket MS-Word)

## Real world examples

- WinCE
- Office automation with internet connection
- Symbian EPOC OS
- Opera browser

## Contact

- [stevec@basistech.com](mailto:stevec@basistech.com)
- [www.basistech.com](http://www.basistech.com)