Developing Unicode-aware Applications in Python

Preparing an application for internationalization (i18n) and localization (l10n)

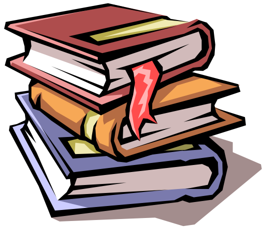
LSM Conference 2005
Dijon, France

Marc-André Lemburg

EGENIX.COM Software GmbH
Germany

Speaker Introduction: Marc-André Lemburg

- CEO eGenix.com and Consultant
  - More than 20 years software experience
  - Diploma in Mathematics
  - Expert in Python, OOP, Web Technologies and Unicode
  - Python Core Developer
  - Python Software Foundation Board Member (2002-04)
  - Contact: mal@egenix.com
- eGenix.com Software GmbH, Germany
  - Founded in 2000
  - Core business:
    - Consulting: helping companies write successful Python software
    - Product design: professional quality Python/Zope developer tools (mxODBC, mxDateTime, mxTextTools, etc.)
  - International customer base



Agenda

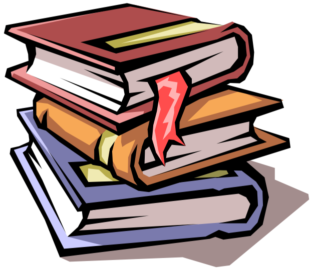
1. Introduction
2. Preparation for Internationalization
3. Adding Translation Support
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

Introduction

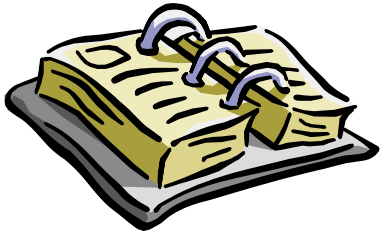
1. Introduction
2. Preparation for Internationalization
3. Adding Translation Support
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

Motivation: Why Unicode ?

- Storing scripts: human readable text data
  - Localization (l10n) and Internationalization (i18n) of software and GUIs
  - Basis for national language and script support
  - Common ground for textual information exchange



The Unicode Consortium Solution



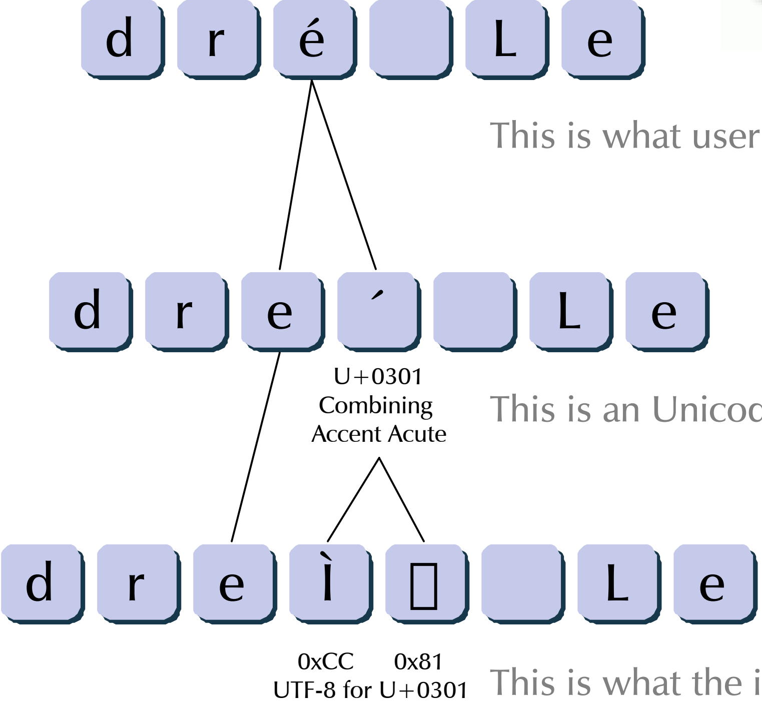
- One encoding for all scripts of the world
- ASCII compatibility (even Latin-1)
- Includes character meta data
  - Case mapping information
  - Numeric conversion
  - Character category information
- Accounts for scripts using different orientations
- Enables sorting and normalization support

Also see the Unicode Consortium web-site at http://www.unicode.org/

Unicode Terminology: What is a Character ?

- Unicode Terminology

  - Graphemes:
  - Code Points:
  - Code Units:



This is what users regard as a character.

This is an Unicode encoding of the string.

This is what the implementation stores (UTF-8).

Unicode Statistics

- Unicode 4.1.0
  - 1,114,112 code points available
  - 97,655 code points assigned
    - 1,273 code point assignments were added in Unicode 4.1.0 compared to Unicode 4.0
  - 70,207 of these are part of a Han subset used for Asian scripts
  - Most assignments in the first 65536 code points (BMP - Basic Multilingual Plane)
- Python supports Unicode version 3.2 (in Python 2.4)

Unicode features included in Python

- Native Unicode Type
  - very efficient
  - performance comparable to strings (sometime even better)
- Large set of built-in codecs
  - to convert between Unicode and various encodings (among other things)
- Unicode code point database
  - information on code point properties
- Partial support for OS based Unicode I/O
  - still in the making

Unicode literals in Python

- Source code encoding
  - Defines the encoding used for the Python source code
  - Must appear in the first two lines of a Python program
  - Format: `# -*- coding: latin-1 -*-`
- Unicode literals
  - String literals prefixed with a small *u*
  - Get converted to a Unicode object
  - Format: `u”this is a latin-1 string (éèàôäöü)”`

Pitfalls in writing Unicode-aware Python applications

- Not all Python modules/extensions expect Unicode
  - UnicodeError (due to ASCII conversion)
  - TypeError (tool expected a string)
  - Work-around: explicit encoding/decoding
- Operating Systems
  - don’t all handle Unicode well
  - Python doesn’t always use their Unicode support
  - Work-around: use ASCII OS-identifiers wherever possible
- Tool-chain:
  - Unicode is still in the process of being adopted – we’re not quite there yet… YMMV

Preparation for Internationalization (i18n)

1. Introduction
2. Preparation for Internationalization
3. Adding Translation Support
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

General principles in preparation for i18n

1. Use Unicode for all text in the application / presentation data
   - Avoid mixing strings and Unicode
2. Use explicit encoding/decoding in all I/O operations
   - Avoid Python's automatic coercion mechanisms
   - Encodings are usually application and locale dependent

I18n approach in Python: Basics

- Choose a default language
- Always define the source code encoding
  - should be suitable for your default language
  - Example: `# -*- coding: latin-1 -*-`
- Always use Unicode literals for all text
  - written in your default language
  - Example: `u”use your preferred default language”`
  - Important:
    These strings will be used as keys to find their own translation

I18n approach in Python: Prepare for automatic translation

- Enclose all literals in a call to a translation function

  translate(u"Save Document")

  translate(u"Save Document", topic=u"Menu")

  _(u"Save Document") (for those who don't like typing ☺)

- Always inline formatting specifiers into literals

  _(u"this will cause ") + many + _(u"translation problems")

  _(u"this is much %s translation friendly") % (more)

- Try not to break literals unnecessarily

  _(u"complete sentences are usually easier to translate…")

  _(u"…than short snippets without context")

Translation Problems

- Strings can have different translations depending on context
  - Use topics (aka domains, categories)

- A single string in one language can have multiple translations in other languages
  - Try to make the string more descriptive, or
  - Add helper context which the translation function then removes again for the default language

- Missing translation ?
  - Fallback to the default language

Adding Translation Support

1. Introduction
2. Preparation for Internationalization
3. **Adding Translation Support**
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

Translation Tools: GNU gettext tool chain

- Python gettext module (Python license)
  - provides translation function
- Many available tools:
  - to extract literals from source code (xgettext)
  - manage translations
  - compile translations for quick lookup
- Problem:
  - limited topic support
  - not context-aware (at least not out of the box)
  - hard to extend

Translation Tools: eGenix approach

- Use a TranslationComponent in the application
  - translations stored in the database
  - provides translation function
  - “knows” what the application is doing: context aware
- String extraction:
  - dynamically at run-time
  - statically, by scanning source code and/or presentation data

Translation Tools: eGenix approach (cont.)

- Managing translations:
  Import/export translations to Excel Unicode CSV files
  - easy to pass to translation studios
  - can include topic information

- Advantages of the approach:
  - context- and topic-aware
  - easily extendable
  - tested and proven in real-life applications

Interoperability

1. Introduction
2. Preparation for Internationalization
3. Adding Translation Support
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

Application Interoperability

- For best interop, use UTF-8 as Unicode transfer format
  - Best supported transfer format
- Avoid UTF-16, if possible
  - Byte ordering issues can be troublesome
- Avoid lossy encodings such as Latin-1, ASCII, etc.

Common Unicode transfer formats

- Browsers
  - UTF-8 (good support on all platforms)
- Text Editors
  - UTF-8 (Joe, Emacs on Unix)
  - UTF-16-LE (Notepad, Word on Windows)
- Excel
  - CSV files: UTF-16-LE
- Terminals / Shells
  - UTF-8

Detecting character sets / encodings

- Very hard problem (in general)

- Some encodings help
  - UTF-16 uses BOMs (byte order marks)
  - UTF-8 sometimes does too

- The application may have enough knowledge to detect the encoding based on the context …
  - … or it may not ☹

Localization (l10n)

1. Introduction
2. Preparation for Internationalization
3. Adding Translation Support
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

General things to consider when localizing (l10n)

- Date formats
  - 2005-07-07 vs. 07.07.2005 vs. 07/07/2005
- Number formats
  - 1.234,567 vs. 1,234.567
- Currency formats
  - $12.34 vs. €12,34 vs. 12.34 MUR
- Translations for varying quantities
  - Singular and plural form: u"%i file(s)"
  - Empty set or zero: u"no files"

GUI considerations

- Text direction: Left-to-right vs. Right-to-left
  - Text
  - Menus
  - Buttons
- Varying sizes of glyphs depending on language
  - e.g. English compared to Japanese
- Accelerator Keys
  - will likely have to depend on the language

Discussion

1. Introduction
2. Preparation for Internationalization
3. Adding Translation Support
4. Translation Tools
5. Interoperability
6. Localization
7. Discussion

Developing Unicode-aware applications in Python

- Questions
  - What is your biggest problem with Unicode ?
  - What tools / features are (still) missing in Python's Unicode support ?

And finally...

Thank you for your time.

Contact

eGenix.com Software, Skills and Services GmbH
Marc-André Lemburg
Pastor-Löh-Str. 48
D-40764 Langenfeld
Germany

eMail:	mal@egenix.com
Phone:	+49 211 9304112
Fax:	+49 211 3005250
Web:	http://www.egenix.com/