UNITED NATIONS ECONOMIC COMMISSION FOR EUROPE

Principles and Guidelines

on Building Multilingual Applications for Official Statistics

UNITED NATIONS

Principles and Guidelines on Building Multilingual Applications for Official Statistics

These Principles and Guidelines were produced by the Conference of European Statisticians Sharing Advisory Board and the UNECE Secretariat, with input and peer review from Participants in the 2011 joint UNECE / Eurostat / OECD meeting on Management of Statistical Information Systems (MSIS). This version was published in February 2012.

I. Introduction

1. The move towards global standardisation in models such as the Generic Statistical Business Process Model (GSBPM) and the Generic Statistical Information Model (GSIM), combined with progress on the development of standards for exchanging data and metadata, has drawn the attention of statistical software providers to the possibilities of exchanging software internationally. This has prompted the question of how we can actively incorporate this possibility into the development of software from its inception.

2. Statistical software is often developed in the national context; multi-lingual support can be an afterthought. Adding support for other languages later is often much more difficult if language is not considered as a key part of the software architecture from the start.

3. However, there is more to building software that will be used in many countries than just focussing on language issues alone. Adhering to best practices in internationalising software means that the possibilities for sharing data within the statistical community are increased. The implementation of these guidelines will increase the portability and reuse of software.

4. The purposes of these guidelines are to provide a review some of the common best practices to develop internationalised software, to highlight some of the resources and common standards in the area and to focus on topics that may specifically apply to statistical software such as the treatment of numbers, formatting, dates, formula treatment, and notation.

II. Definitions

Internationalisation and Localisation

5. The process of developing software for use in multiple languages, cultures and countries can be viewed as two separate processes, those of internationalisation (I18N) and localisation (L10N). Internationalisation deals with the process of designing software so that it can accommodate changing environments without changes to the code. Localisation deals with the more specific case of designing for example for a country, language and or region. So while internationalisation

concerns itself with providing the system to allow the use of more than one language, localisation would instead involve the application of this such as translation etc.

Locales

6. A locale is a collection of user preferences applicable to a specific language country and or culture. Locales identifiers usually consist of a language, often combined with a country and occasionally with a further parameter specifying the code set and modifier; for example en_GB is the locale identified for English for the UK. This means a differentiation can be made between English (US) and English (UK). The locale consists of a number of elements including for example the name and ISO identifier of the language, the currency, sorting requirements, numeric preferences such as thousands separators , the calendars to be used and other elements such as text direction (left-to-right or right-to-left, horizontal or vertical) etc.[1]

III. Principles

Principle 1: Software should be multilingual by design

7. Software should preferably be designed and tested to implement multilingual application rather than retrofitted. It is recommended that the requirement for software to facilitate the use of more than one language should be included in the initial software requirements. It is easier to implement internationalisation if it is incorporated from the beginning. When it is included in requirements, it is developed and tested throughout the development process. This will provide a higher quality product than if the language requirements are added on at the end of the process.

8. Requirements specifications should include whether more than one language should be stored, presented and output at one time, for example in a bilingual country; this has an impact on how multilingual applications are implemented.

Principle 2: Multilingual applications should support additional languages without reengineering

9. Where an application has been designed to allow the use of more than one language, it should facilitate the introduction of new languages without having to be extensively rewritten. The program should be designed where possible with the ability to adapt to add new languages. The storage of the user interface elements and application data should also facilitate this. Following the guidelines in this document will highlight some of the issues to be aware of.

10. It is recommended to provide multi lingual systems if possible with one set of binaries, so that the code is consistent and to reduce the costs of installation, support etc. Only one version then needs to be maintained. This however, poses a

[1] http://www.jbase.com/new/support/41docs/jBASE%20internationalization.pdf

challenge in that display elements need to be designed to accommodate varying languages and lengths. This may also increase the size of the application.

11. Where this is possible it will reduce lag time in providing international versions and improves the quality of the product overall.

Principle 3: The User Interface should always be separated from the code

12. Separating the User interface (UI) from the code makes the rest of the application effectively language neutral, which has a number of benefits.

- Addition of new languages or changes to the user interface does not require recompiling or reprogramming of the code.
- Translators can work without needing knowledge of the code or logic of the program.

13. One example of this separation would be using string variables rather than string constants within the code. The strings variables are then loaded based on the selected locale or language.

Principle 4: Identify and adapt all elements of the user interface

14. When talking about the user interface we should keep in mind that this also includes not only the text used in the user interface. Languages changes can also affect the layout of the user interface, such as the buttons, the size of the application window etc.

15. When designing the interface the designer needs to accommodate possible changes to the text lengths in the design of the forms. Word lengths vary between languages. This particularly affects labels, as the variability of text length in different languages is most pronounced in short phrases or text . As more words are included in sentences the difference tends to average out over the longer text. For example ‘Click here’ (10 letters) translates to ‘Klicken Sie hier’ in German (16 letters).

16. A guide to the amount of space required for translated text is reproduced below from Oracle’s “Understanding Application Development Guidelines”

Number of English Characters	***Additional Space Required***
1 character	400 percent or 4 characters
2—10 characters	101—200 percent
11—20 characters	81—100 percent
21—30 characters	61—80 percent
31—70 characters	31—40 percent
More than 70 characters	30 percent

17. Room for expansion needs to be included in menus, labels and dialogs. All of these elements need to be re-tested after translation. It is recommended that you shouldn’t overcrowd text in forms and applications. This allows translated elements

to display effectively. Relative sizing for elements such as label boxes, containers etc. should be used rather than fixed sizing where possible.

18. Icons and graphics should also be recognised as part of the user interface; to allow translation of the interface you should restrict or avoid the use of embedded text in icons and graphics. Any text used should be drawn from string variables.

19. The icons and graphics should undergo similar checking and translation as the string text to identify any possible misinterpretations. The interpretation of icons and graphics differs between cultures. Pointed fingers for example can be offensive in some cultures. Colours can have strong positive or negative connotations in different cultures. These elements should be used with sensitivity.

20. Accelerator keys, menu shortcuts etc. are all elements of the user interface that also need to be adapted. Shortcuts should be adapted for different languages; this requires the storing of the shortcut keys as part of the user interface. Shortcut keys also need to be accessible from different keyboards. This requires checking keyboard layouts for different locales.

21. The entire user interface should at some stage of testing be reviewed in the translation work and made available to the localisation team.

Principle 5: Plan the storage and management of multilingual user interface elements

22. The user interface elements should be maintained in an accessible and trackable format. This can be in for example a resource file format, a database table etc. The storage should include metadata regarding the interface elements such as comments, context, IDs etc. This can be used to communicate with translators, for example where certain text is used, when it appears, what it conveys and also simplifies conversations between translators and developers for clarification.

23. IDs allow the tracking of the translatable elements and also improve the possibility of their reuse between versions or separate applications. This allows the standardisation of the translations and can reduce translation costs if the same text can be reused. When working with XML for example it is recommended by the Internationalization Tag Set (ITS), that translatable text should be stored in elements rather than attributes to allow unique id's, comments etc.

24. The string elements required for different languages can differ in length and therefore size; ensure that storage elements can expand to accommodate strings in different languages. [2]

25. The order of messages can change between languages so that messages should be stored as complete sentences rather than reconstructed from keywords. Similarly avoid using ordinals with numbers, these are not always easy to translate. For example, instead of "The 1st record of the table is", use "Record 1:"

[2] See Guidelines 8 and 12 for further discussion on saving multilingual data

26. Text that contains variable data can pose a problem. For example: “The table has 10,000 *records”* translates in Romanian to "Tabelul are 10.000 *de înregistrări*" with a change in the positioning variable between the beginning and end text. The sentence itself could be simplified for example to 10,000 records -> 10,000 înregistrări. or by the use of placeholders in the text "The table has {1} records", where {1} would be replaced by the value (10 000) during runtime.

27. A default language should be specified in all cases so that, at a minimum, some message is displayed in the event of missing text or images. The user should have some notification that the element is missing in their requested language. The missing translation should be logged as a system error

Principle 6: Decide the best way to select the locale

28. The standard locale implementation should attempt to select the most appropriate locale for the user initially. This could use the system or user default locale of the operating system, the characters detected in a document or a standard default option as appropriate.

29. Users must then be given the opportunity to select and change their locale. For example a default language may be set by the country of the ip addresses but most international websites recognise that a user may prefer a different language and allow the default to be changed.

30. Remember the locale selected and continue to use it during the session. Where a user can be identified, save the preferred locale and use the same preference in future. With a website for example, cookies can be used to default the preferred language in future sessions.

31. Ideally a user should be able to change locales at any stage during run time so that the application does not need to be restarted to change locales. However changing locales after start-up may mean additional server loads (querying more than once) and this should be taken into account when specifying.

32. If locale change is allowed at a later stage then existing data should be retained or if not possible, a warning of data loss must be given. The opportunity to cancel the locale change before refreshing the entered data should be allowed. Ideally the page status should be maintained on locale updates;
Language selection options should use their native names for example, English, French or their ISO abbreviations, en, fr, de etc. Do not use national flags to identify languages.

33. Users should be able to select their own input method, such as the keyboard layout, to allow them to enter characters using the key combinations that they are used to.

Principle 7: Presentation of data should follow the customs of the locale

34. Presentation of data differs between locales. Some of most common areas where differences occur are detailed below. Development of software should be aware of and take into account possible differences. Care should be taken that the application does not rely on how things are presented. For example: refusing to accept Zip code formats of different locales. To handle differences in presentation the appropriate format as defined by the locale should be applied. Where multiple formats are available users preferences should be saved. Outputs should be able to display the required character sets.

Examples of Locale differences in presentation formats

- Numbers:
  - Number formats such as the position of decimal separators etc differ
  - Ordinality differs between languages e.g. 1st etc
  - billion vs. trillion in the long and short scale of numbers
  - How many significant digits should be displayed
- Dates: Different format types, different calendars used, different working days,
- Measuring Units:
  - Imperial vs. metric
  - Currencies e.g. symbols , placement of currency identifier
- Text Input and Layout: Text can be displayed in different directions e.g. right to left
- Date/Time Formatting
  - Calendars Gregorian, Thai etc.
  - Time Zones
  - Working days, e.g. Start of the week
  - Formats e.g. US. MM/DD/YY
- Paper size:
  - US paper size differs from European paper sizes.
  - Output should not be limited to a ‘standard’ paper sizes
- Name formats, titles used e.g. Mr, Mme; layout of names, forename first or last name.
- Legal requirements
  - Determine whether locale specific trademarks, logos or branding are used and find out whether they can or should be used in other locales
  - Conforms with regulations
- Address & telephone formats This is of relevance for many statistical applications, special attention should be paid to
  - The ability to enter addresses in languages as desired.
  - Ordering of address fields e.g. Street name first etc.
  - The availability and format of postcodes.
  - Country names: Locales will supply the lists of countries should be used for countries and geographical areas

Principle 8: Processing and saving of data should follow the customs of the locale

35. Processing the data within your software should take account of the possible differences between locales. In order to do that it should be noted that the following functions that may be used in processing can differ between locales

- Rounding
    - Differences exist for example
        - How many significant digits should be used? e.g. currency rounding
        - Where data are not compared between locales, use the locale defined rounding
        - If data are compared then standard rounding should be used.
- Strings [3]
    - Input and saving
        - Can all characters be input and saved?
        - Are different keyboard layouts facilitated?
        - Are you relying on single characters for separate keystrokes that may not be available on all keyboards?
    - String comparison:
        - The search and sort order used should be that of the current user preferences.
        - You should use locale dependent string sorting functions where available when comparing locale dependent text (such as lstrcomp). Standard String comparison functions may not take account of localised sort orders.
        - Searching and other string functions should be by character and not by byte size due to variable character byte sizes.
        - Where text is not locale dependent use non locale dependent string functions
    - String manipulation
        - Line breaks, spacing etc differ, don't rely on a standard break in your processing.
    - String sorting
        - Sorts differ between different cultures and alphabets.
        - There may be more than one sort order available, e.g. German phone book order.
        - Unicode sort order does not match linguistic sort order or expected binary sort ordering. Do not rely on a assumed sort order.
        - Decide whether searches over data in multiple languages should be allowed.
            - This may be required in multilingual countries such as Switzerland
            - A dominant search order should be defined.
        - How should accented characters be handled in searches?

[3] See references section: 'Strings' for links to more detailed string discussions

        - Be aware of possible differences with database sort orders and the collations used
    - Currencies
        - Store the currency amount along with currency identifiers.
        - Should multiple currencies be able to be stored?
    - Measurement units
        - Store the measurement unit along with the measurement.

36. Most importantly watch out for hidden assumptions about how things work.

Principle 9: Reuse standards such as program libraries for localisation

37. Standard locales should be used in preference to defining customised settings. Standard locale information can be accessed through many development libraries and resources. The Unicode Common Locale Data Repository is an example of a comprehensive library of downloadable locale information (see references for further details).

38. Most development software has localisation routines available for you to access locale information; for example setlocale in Microsoft Visual studio; International Components for Unicode (ICU) for java and C++ development.

Principle 10: Include estimates for provision of multilingual support in estimates of development costs

39. It will take more time to develop and test multilingual software. If translations are being included with the software then allow for the translation time in the software development plans. Include the translation effort not only of the text but also checking of the user interface. As before ,check that text strings are displaying correctly, are they being cut off, are the icons and graphics acceptable?

Principle 11: Include documentation as part of localisation efforts

40. User documentation should be available in the language of the locale. This should be thought of as part of the user interface. This translation can be included in the localisation effort. As a minimum, user manuals should translated, administrator files should be included if possible.

41. In addition other textual objects such as help files should be included in the localisation efforts along with system messages.

Principle 12: Understand Unicode and use it where possible

42. Internally characters are saved and stored on computers as binary numbers, the mapping of these numbers to their character values is known as a character set. ASCII is one of the original standard character sets. Initially characters were stored

in 7 bits of memory and were intended to represent the standard Latin alphabet, numbers along with punctuation. With 2^7=128 available mappings.

43. In this way the character ‘a’ could be represented as below

Binary	Ascii code	Character
110 0001	97	a

44. To represent the standard English language character set codes were defined for a set from 1 to 127. Codes below 32 used for system processes, ASCII codes between 32 and 127 were reserved for the traditional Latin character set of 94 printable characters along with a space character.

45. The introduction of 8 bits for storage of characters meant that up to 255 different characters could then be stored, including the original code mappings this meant that the codes of 128 to 255 were ‘free’. This allowed an extension of the character set to include accented characters or other mappings as required. Codes above 128 were then implemented in different ways in different countries. This freedom meant that text sent using the ASCII character set could have different representations in different locales, with the same binary code assigned to multiple characters. Code sets were developed to define and communicate how the ASCII values above 128 were used e.g. ISO Latin 5 (’Turkish’). However 8 bits and 255 characters were still insufficient for some languages such as Chinese, Japanese, Korean etc.

46. Unicode and ISO 10646 (also known as UCS for Universal Character Set) were developed as an international standard to allow consistent representation and treatment of all characters. Each character has a defined number or ‘code point’, for example a code of U+0021 refers to the character exclamation mark ‘!’. Unicode can currently facilitate over 1 million characters with over 100,000 assigned in the current version. The first 128 characters are compatible with the ASCII character set.

47. The introduction and adoption of Unicode allowed the separation of the coding of characters from their storage. The code points could be stored using different encoding systems.

48. The most common encoding systems include UTF-8, UTF-16, and UTF-32. In deciding which encoding system to use it is essential to consider limitations imposed by memory, storage and the need for backward compatibility. A brief overview of the is provided with further information on comparisons between them available on Wikipedia at http://en.wikipedia.org/wiki/Comparison_of_Unicode_encodings

49. In UTF-8, every code point from 0-127 is stored as 1 byte. Code points with a value of 128 and above are stored using up to 6 bytes. This means that for the first 128 characters the ASCII character set maps to the UTF-8. The first byte in a multibyte character indicates how many bytes are used for the character. This is the default used for XML.

50. UTF-16 encoding is the standard used for windows. It uses two bytes per character as standard. Most UNICODE characters are encoded by their

codepoints.UTF-16 is popular in areas using DCBS (double character byte size) such as China, Japan. UTF-16 is used by Java and Windows.

51. UTF-32 uses four bytes per character.

52. Unicode also allows the use of older encoding systems; where a code does not exist in the old encoding system a default missing value is displayed; this eliminates the problem of characters being swapped in different coding systems. At a minimum however, in order to represent text correctly the encoding text must be specified.

Using Unicode:

53. To use UNICODE in your applications:

- Use compatible code sets and communicate this code set. You should be able to read data generated in different code sets and process the data correctly; For example:
  - In emails the encoding system should be given in the header as for example **Content-Type: text/plain; charset="UTF-8"**
  - With HTML pages the encoding should always be given in the Meta tag in the head <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /> Since December 2007 **UTF-8** has overtaken ASCII and other code sets to become the most commonly used encoding on web pages.
- Use Unicode compatible data types and functions where possible

Saving data in Unicode format:

54. Data should be stored in Unicode data types; for example in SQL server uses nchar, nvarchar, and nvarchar(max); instead of their non-Unicode equivalents, char, varchar, and text. These storage type and functions are able to allow wider characters. Data should be saved or converted as appropriate in a Unicode format.

55. Care should be taken when data stored in non Unicode data types is converted to Unicode data types as this could cause issues with missing data, this should be flagged as an error when it is detected.

56. Use Unicode compatible functions where available.

57. There can be some performance cost and differences with Unicode string functions, these should not be prohibitive but for intensive text operations it may require optimisation[4].

[4] http://support.microsoft.com/kb/322112 **Comparing SQL collations to Windows collations** Because the comparison rules for non-Unicode and Unicode data are different, when you use a SQL collation you might see different results for comparisons of the same characters, depending on the underlying data type.

Principle 13: Consider what fonts to use and how to use them

58. There are a limited set of Unicode fonts which aim to represent most if not all of the Unicode character set, these include Arial Unicode MS. Generally, it is recommended to use the most appropriate fonts for the locale and language in preference to UNICODE fonts. A font should ideally be chosen that also supports a broad range of accented characters while satisfying design considerations. Simple fonts should be preferred to a script or calligraphic type font.

59. Some options for selecting fonts are
- Using logical font names: e.g. sans serif
- Defining the font by name
- Using true type fonts
- Bundling fonts with your application.

60. A fallback font should be defined.

61. Formatting elements differ between languages, for example bold is commonly used for emphasis in many scripts but for example in Kanji and other scripts it may cause problems with legibility. Other emphasis techniques such as underlining may be used instead.

IV. Conclusion

62. The importance of providing software that can be used in more than one language or culture has increased. The last decade the growth of the internet has fuelled the sharing of software. To gain from future developments within the statistical community software should be developed with an expectation that it will be used in more than one environment.

63. This document has outlined some of the main recommendations for developing software that can be adapted to multilingual use. It is not intended to provide an exhaustive list of issues that may be encountered. It will provide grounding in common problems encountered and an awareness of how to avoid or counter them.

64. The guidelines above have been drawn from a review of documentation of both practical examples of internationalising software and research. It is recognised that implementing each guideline in every application may not be feasible but they are intended to provide an overview of best practice to work towards.

V. Resources

Guides:

- IBM have also published a comprehensive guide to developing international software that provides an in-depth review of issues involved in developing software for multi-culture use available at http://www-01.ibm.com/software/globalization/guidelines/
- Globalization Step-by-Step: http://msdn.microsoft.com/en-us/goglobal/bb688121
- Get World-Ready Microsoft: http://msdn.microsoft.com/en-us/goglobal/bb895995.aspx
- Wikipedia: http://en.wikipedia.org/wiki/Internationalization_and_localization
- jBASE Internationalization Publication detailing internationalisation efforts for Jbase software http://www.jbase.com/new/support/41docs/jBASE%20internationalization.pdf
- Best practices for XML localisation: http://www.w3.org/TR/xml-i18n-bp/
- UTF-8 and Unicode FAQ for Unix/Linux: http://www.cl.cam.ac.uk/~mgk25/unicode.html
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets : An easy to understand guide to Unicode for programmers http://www.joelonsoftware.com/articles/Unicode.html
- Description of basic concepts for internationalization, how to write internationalized software, and how to modify and internationalize software: http://www.debian.org/doc/manuals/intro-i18n/
- Key challenges in Software internationalisation http://www.acs.org.au/documents/public/crpit/CRPITV32Hogan.pdf

Strings:

- Microsoft Sorting and String Comparison: http://msdn.microsoft.com/en-us/goglobal/bb688122
- Oracle guide: Linguistic Sorting and String Searching http://download.oracle.com/docs/cd/B19306_01/server.102/b14225/ch5lingsort.htm#NLSPG005

Checklists:

- Microsoft Win32 Internationalization Checklist http://msdn.microsoft.com/en-us/library/cc194756.aspx
- Oracle Checklist for localisation of software http://developers.sun.com/dev/gadc/i18ntesting/checklists/allquestions/allquestions.html
- Guidelines, Checklists, and Resources http://www.i18nguy.com/guidelines.html

Common libraries and software resources:

- CLDR: Unicode Common Locale Data Repository: The Unicode CLDR provides a standardised repository of locale data in xml format. http://cldr.unicode.org/

- GNU C Library: The GNU C Library, commonly known as glibc, is the C standard library released by the GNU Project; http://www.gnu.org/software/libc/ For example 7.6 Accessing Locale Information: http://www.gnu.org/s/libc/manual/html_node/Locale-Information.html
- The Java platform :java.util.Locale http://java.sun.com/developer/technicalArticles/J2SE/locale/
- ICU: “a mature, widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications” http://userguide.icu-project.org/i18n

Standards:

- Unicode: The Unicode Consortium is a non-profit organization devoted to developing, maintaining, and promoting software internationalization standards and data, particularly the Unicode Standard, which specifies the representation of text in all modern software products and standards; http://www.unicode.org
- The current published Unicode standard is version 6 which contains 109,000 characters http://www.unicode.org/versions/Unicode6.0.0/ch01.pdf
- Internationalization Core Working Group Home Page: Provides internationalization advice to other groups developing Web standards http://www.w3.org/International/core/
- Wikipedia article on ASCII: http://en.wikipedia.org/wiki/ASCII
- Wikipedia article on UNICODE: http://en.wikipedia.org/wiki/Unicode

Guides by national authorities

- BILINGUAL SOFTWARE GUIDELINES AND STANDARDS- Welsh Language Board http://www.byig-wlb.org.uk/english/publications/publications/3963.pdf
- Building bilingual applications at StatCan (Karen Doherty, Statistics Canada) http://www1.unece.org/stat/platform/download/attachments/22478904/issue+4.pdf?version=1

Examples:

- Lessons Learned From Internationalizing a Web Site Accessibility Evaluator http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.2698&rep=rep1&type=pdf

UNECE, Geneva, 2012