CHAPTER 1

Globalization with XML

In This Chapter:

Internationalization is the process of developing a product in such a way that it works with data in different languages and can be adapted to various target markets without engineering changes.

Localization is the subsequent process of translating and adapting a product to a given market's cultural conventions.

The global strategy for any given product or service must include at least an internationalization part and almost always some localization. With the growing role XML plays in the management of data, it has become a significant component in both areas.

Although documentation and Web sites remain obviously important domains for XML applications, the utilization of the markup language goes beyond that today. You can now see XML formats for user interfaces, data transfer layers, repository format, database export/import mechanisms, graphics, and many other applications.

In addition, XML is increasingly used during the localization process itself, regardless of whether the projects have XML components.

This chapter provides a close look at what are called markup languages, at what internationalization and localization mean and their importance in the development cycle of a product, and at some of the challenges XML material and languages offer.

Markup Languages

The term *markup language* covers two main notions that are often confused. A good understanding of these concepts can go a long way toward clarifying their respective issues.

- **Metalanguages**—A metalanguage is a set of syntactic rules that can be used to describe different formats. Examples of metalanguages are SGML and XML. With these two metalanguages the description of a format is usually done through a DTD (Document Type Definition).
- **Document Types**—The second aspect of markup languages is the document type: a format devised to serve a specific purpose. You can see it as an application written in one of the metalanguages. Examples of such applications are TEI, HTML, OpenTag, XSL, and MARTIF. In XML context, document types are often called *vocabularies*.

Figure 1.1 illustrates this framework: Both metalanguages are used to define different document types.

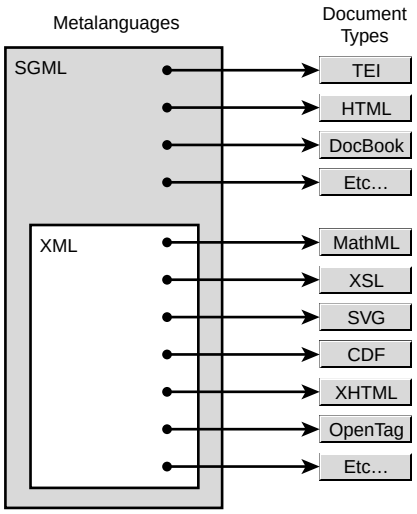

FIGURE 1.1

The relationships between markup languages.

SGML (Standard Generalized Markup Language) was created in the mid ’70s and became an ISO standard (ISO 8879) in 1984. As the importance of the Internet grew during the last decade, it became clear that a lighter and less complex version of SGML would be very useful. Several initiatives of simplified SGML were sketched:

SGML-Lite, PFSGML (Poor-Folk's SGML), SO (SGML Online), or MGML (Minimal Generalized Markup Language) are some of them. All these efforts finally resulted in the emergence of XML.

XML, or eXtensible Markup Language, is an international standard developed by the World Wide Web Consortium (W3C). The version 1.0 specifications were released in February 1998, and a second edition of version 1.0 was published in October 2000. One of the big differences between SGML and XML is that an XML document is not necessarily associated with a DTD: You can parse XML files without knowledge of how the elements are structured. This simplifies greatly the development of XML-enabled tools.

On the document types side, some formats are more known than others: For example, HTML (Hypertext Markup Language) is widely used. Although it is merely a document type like any other, it is often mistakenly presented as a sibling of SGML or XML.

The recent XHTML specification is the W3C's recommendation for the latest version of HTML. XHTML is an XML application, just as HTML was an SGML one.

Key Components of the Development Cycle

Internationalization and localization are key parts of the development cycle of your product or service. They should be clearly identified as important components of the production process.

The different areas where internationalization and localization are needed in the project must be clearly identified and the relevant personnel should be assigned to them. Someone must *own* these aspects of the project and they should be taken into account in your schedules and budgets.

Make no mistake: Failure to think about localization ahead of time and to take the necessary steps to develop an internationalized product will always have financial consequences later. This is true not only for software applications, but also for documentation, Web sites, help systems, or any other component of your product.

Three areas of action are involved in internationalization or localization:

- **Development and authoring**—How the source material is created and updated.
- **Pre- and post-processing of the content**—How the localizable parts are routed, leveraged, tested, and adjusted in any ways before or after translation.
- **Linguistic work**—How the content is translated, edited, and proofed and also how the terminology is established, maintained, and verified.

Figure 1.2 shows the interactions of these three different areas. Obviously, the various tasks are often tightly related. For example, establishing the terminology is likely to occur during the development, or some of the pre-processing can be taken care of at the development stage by implementing appropriate methodology or using dedicated tools.

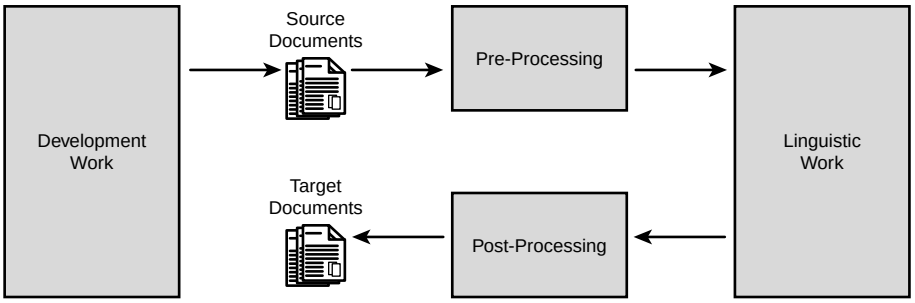

FIGURE 1.2

The relationships of the three main areas of multilingual content development.

The work can be done by different departments within the same organization, or by different vendors. As a general rule, the fewer the steps between the linguistic work and the authoring, the less costly the localization will be.

Regardless of whether you use external resources to do part or all of these tasks, the same guidelines apply.

Internationalization

Often, internationalization is seen as an activity related to software engineering: making sure double-byte strings are handled properly, that all translatable text is externalized, and so forth. In reality, internationalization goes much beyond those simple tasks and affects every single piece of your projects. For example, preparing for localization is closely tied to content development. Keep in mind that the biggest chunk of your budget will almost always be the cost linked to documentation. That is where you can realize substantial savings if you optimize the source material and the process to take into account its multilingual nature.

Internationalizing your project involves two sequential activities: enabling and preparing for localization.

Enabling

The most important facet of internationalization is to make sure the data contained in your document can be processed correctly regardless of its language or the script it uses.

In XML this means your vocabulary has elements describing the intent rather than the rendering, that localizable content is easily identified and lends itself well to the use of translation tools and surrounding processes.

If you are doing any transformation or rendering of the data, any templates and style sheets you might use take into account locale-specific requirements.

The following list enumerates some of the aspects you should be able to address after enabling is done:

- Text with extended characters (that is, non-ASCII characters) can be rendered appropriately, including ligatures, composite characters, writing direction, and so forth.
- Users can input text in different languages correctly and the result is processed and stored without corruptions.
- Numbered lists are generated with the correct digits.
- Ordered lists or tables are sorted according to the rules specific to the locale.
- The different languages in multilingual documents are identified properly and consistently.
- Text is wrapped adequately even if the language does not use spaces to separate words.
- The content can be searched or indexed correctly regardless of its language.
- Any automated text, such as quotation marks or captions, is generated according to the locale of the text.
- Numbers, dates, and time are formatted according to the rules used for the given locale.
- Whenever possible you use a system of IDs that will make the reuse of previously translated content easy.
- Whenever it makes sense you use XML features to reduce the repetitions of translatable parts. For example, you could use external entities for frequently used well-defined paragraphs. Do not go overboard though.
- Formatting is marked up in a generic way and different locale-specific styles can be applied easily.

Thanks to the work of many people at the W3C, XML is already enabled: It uses ISO-10646/Unicode, provides all the mechanisms to identify languages and encodings, and offers all the mechanisms necessary to handle non-Roman languages.

Nevertheless, you still must make sure these mechanisms are used correctly to create XML documents that can be seamlessly localized.

Preparing for Localization

The second aspect of internationalization, which assumes that enabling has been done already, is to make sure that your content can be localized without modification of its structural core or its setup.

In addition, you want to ensure that the work can be done with the maximum efficiency to lower costs and obtain rapid turnaround time.

The following list enumerates some of the aspects to address:

- Non-translatable parts within translatable documents are easily identifiable.
- Terminology is established and maintained.
- Any special requirements, such as maximum length or forced use of a subset of characters, are documented.
- The list of components that need to be localized is created and updated, and you have a process to assemble these components into a Localization Kit.
- You have methods in place to test the localized versions of your XML data. Any automated testing procedure can be used on translated data as well.
- Any modifications of the content of the source language that is done automatically can be reproduced for the target language as well (for example, creation of an index).
- The directory structure of your data takes into account the localized components.
- You have designated people responsible for the support of the localization process, for answering questions, for dispatching feedback from the localizer to the developers and authors, and so forth.

All these details can make a significant difference when localization occurs. Always keep in mind that ultimately the complexity of your source material will be multiplied by the number of target languages.

Localization

After your material has been internationalized and is ready, you can move to the localization stage.

Whether the localization is done by a localization vendor or not, the process involves very similar steps. The complexity and details of the process obviously depend largely on the type of XML documents to work on, but it usually involves, in one form or another, the following activities:

- Analysis
- Preparation
- Translation and Editing
- Post-processing
- Verification and QA

In some cases, for example with online translation or working from a content-management system, the steps will be more integrated.

More in-depth discussions on the different methods to translate XML files are in Chapters 13, “Localization Kits,” 14, “XML-Enabled Translation Tools,” 15, “Online Translation,” and 16, “Using XML to Localize.” The following notes are a simple summary of a possible scenario.

Analysis

After receiving the source material, the first thing a localizer usually does is to verify it. In the case of XML files this means to make sure they are at least well formed. If the files come with a DTD or a schema, you probably want to make sure they are also valid.

Hopefully the localizer has already had a sample of the XML files (that is, at the bid stage), knows what to expect, and has a solution for the localization process.

During this phase, the project is scoped, word counts and other statistical data are gathered, and a timetable can be set up.

Preparation

Preparation is about packaging the files for the linguists. A project seldom comes with XML documents only; in many cases work must be done to prepare the translation for entities, files, graphics, and other additional source files.

If the source XML documents are for one reason or another not supported by the tools, they might have to be pre-processed into a different format, for example extracted into a temporary format such as XLIFF (XML Localisation Interchange File Format) or RTF, where the translatable text is separated from the markup codes.

Preparation can also involve generating leveraged files or preparing translation memories, as well as putting together reference material such as glossaries or assembling context information.

This is also one of the times when people involved with the product in the target markets should be included: They provide approval of the terminology, evaluation of the stylistic guidelines for the translation, and other information specific to the targeted audience.

Translation and Editing

After they are packaged, the files are given to the linguists. The translation and editing is often done by contractors working off-site, so usually some level of coordination as well as FTP, e-mails, or other communications means are involved.

Regardless of what tool is used for translation, having a way to see the preview output of the XML source goes a long way in helping the linguists to provide a good quality translation.

If any translation memories are available, tools can be used to re-cycle interactively the segments of text that have already been translated.

Post-Processing

When the main linguistic steps are completed, the translated material comes back to whomever prepared it and any post-processing work is done if it is necessary (that is, reverting from the format used for translation back to the original XML).

Verification and QA

Depending on the type of XML material (documents, UI, help, database files, and so forth), the verification phase might be involved. For example, in some cases, linguists would look again at the translated text in the running environment that was not reproducible during translation and editing. In others, the files would be processed to generate various outputs that would be proofed.

Then, after a final quality assurance step, the deliverable XML files are ready to be sent back to the production stage.

XML Challenges

XML is flexible and offers a clear set of construction rules. A growing number of parsers and enabled tools are now available, and a flourishing companion technology

(for example, XSL, XPath, or XLink) is rapidly growing. All this has made XML an ideal medium with which to manipulate, store, transport, and exchange all sorts of data, not only on the Internet, but in any environment.

Types of Data

XML is used in a broad range of areas: for example, as a system to store your documentation, as a repository for corporate data, or as an interface for applications. Table 1.1 shows a few examples of XML applications.

TABLE 1.1

A Few Examples of XML Applications

Document Type	*Domain of Application*
SOAP	Capsule to transfer data and executable code across different systems
XHTML	Online documents served up over the Internet
XUL	Language to describe user interface: menus, dialog boxes, forms, and so forth
VoiceXML	Programming language for voice systems
SVG	Coding of text and vector-based graphics
TMX	Interchange format for translation memories
XML Query	Language to query Web documents
Open eBook	Format for electronic book presentation
XTM	Topic maps description
Wf-XML	Language to exchange information between workflow management systems
OMF	Weather observation definition format
CML	Markup language for chemical data

In addition to new formats created since XML has become available, many older SGML content types are now adapted to be XML applications. In localization this diversity of vocabularies comes with its share of problems: Each type of document raises different kinds of challenges.

Patchwork Documents

One of the most powerful features offered by XML is the namespaces mechanism. It enables you to mix elements of different vocabularies within the same document.

For example, Listing 1.1 shows a document that uses three different namespaces. The main one is a catalog entry for which the namespace URI is `urn:MacMillan:Sams:Catalog_v1`. The document includes a `<number>` element borrowed from a different vocabulary (identified as `urn:gov:book:isbn`) as well as some HTML markup (mapped to `http://www.w3.org/TR/REC-html40`).

LISTING 1.1

`Namespaces1.xml`—Example of a Document with Elements from Different Document Types

```
<?xml version="1.0" ?>
<book xmlns="urn:MacMillan:Sams:Catalog_v1"
      xmlns:isbn="urn:gov:book:isbn"
      xmlns:html="http://www.w3.org/TR/REC-html40">
 <title>XML Internationalization and Localization</title>
 <isbn:number>0-672-32096-7</isbn:number>
 <orig-lang>en-us<orig-lang>
 <descr>
  <html:P>
   A practical guide on how to <html:I>internationalize</html:I> and
   <html:I>localize</html:I> XML applications.
  </html:P>
 </descr>
</book>
```

This system provides the opportunity to reuse in your own vocabulary markups that already have been defined. It brings, in some regards, an “object-oriented” dimension to XML.

When localizing such documents, you must make sure that tools support namespaces correctly. The only part that is fixed in the namespace syntax is the URI to which the prefix is mapped. In the example the prefixes `isbn` and `html` could have been named anything else, they are just a shorthand notation.

Another issue appears when leveraging translation. Because the element notation can vary depending on the context, the tags that are within translation memory segments can appear different while they are really identical. This can prevent exact matches from being found when they should.

For instance, the previous document could be also coded as shown in Listing 1.2 and the description text ends up with a different syntax for the same HTML tags.

LISTING 1.2

`Namespaces2.xml`—An Alternate Notation

```
<?xml version="1.0" ?>
<book xmlns="urn:MacMillan:Sams:Catalog_v1"
      xmlns:isbn="urn:gov:book:isbn">
 <title>XML Internationalization and Localization</title>
 <isbn:number>0-672-32096-7</isbn:number>
 <orig-lang>en-us</orig-lang>
 <descr>
  <P xmlns:html="http://www.w3.org/TR/REC-html40">
   A practical guide on how to <I>internationalize</I> and
   <I>localize</I> XML applications.
  </P>
 </descr>
</book>
```

The same problem arises when the prefixes are named differently in separate documents. Note that this can happen only if the documents are not validated against a DTD where unique prefixes would be declared.

The simplest way to solve these cases is to implement leveraging methods that truly abstract the way they store information about inline codes.

On-Demand Documents

New ways to view documents came with the Internet. Rather than being considered final outputs, documents are often built on-the-fly, when requested by a user. Maintaining and updating the information can be done much more efficiently.

The first problem from a localization viewpoint is that such documents are quite similar to compiled programs: Not only is the source not the final output, but the output can be made of various pieces, not necessarily in the same format or at the same location. Consequently, the translators cannot see the context of the text to localize, a restriction that can affect the speed and the quality of their work.

For example, a heading can be translated differently or have different capitalization rules if it appears in a manual or in the index of a help file.

The second issue is that the boundaries between data content and code content become very fuzzy: Embedded scripts in different languages, queries, external references and many other code-related statements can be found between paragraphs of simple text. In some cases (for example, XSL or VoiceXML), it is difficult to decide whether a file should be treated as a source code file or a text document.

Here again the tools and methods must be adjusted to deal with the wide breadth of domains where XML is used and the different types of content it stores.

Single-Source Documents

An aspect related to on-demand documents is the capability to use the same source to generate different outputs, for example a dynamic Web page, a PDF file, and an online help topic.

In this scenario the translators might have to deal with several partial or nonexistent contexts.

On the authoring side the design and organization for multiple outputs are often quite complex and should absolutely include thoughts about the localized versions of the different generated materials. Whatever steps are used to create the final outputs must be reproducible, without any addition, for all targeted languages. An oversight of this issue is most likely to increase the cost of the localization.

Language Challenges

Now that we have seen the main steps involved in globalization and what type of material we will have to implement them, it is time to look at the different challenges that the use of different languages brings.

Languages are usually divided by families: sets of different branches that have evolved from a main original root. In the world of computing, languages are most often grouped based on rendering properties that are required for each language: what character sets they use, writing direction, and so forth. Some present only a few difficulties whereas others offer more difficult and unique problems.

Beyond the problems of rendering text, many additional issues make each language special: grammatical rules for gender, accord of the adjectives, noun and verb inflexions, cultural issues for graphics, abbreviations, collation, and so forth.

Character Sets

There are different ways to represent speech in a written form: Alphabets, ideographs, and syllabic notations are the main ones.

An alphabet is a finite set of characters (letters), combined to form the representation of sounds or grammatical information (for example, the Latin, or Roman, alphabet). In some cases each letter can have different forms depending on the context (its position in the word, the previous or next letter, the semantic, and so forth). Uppercase and lowercase are examples of such forms, but some scripts like Arabic go beyond that and use even more forms.

Syllabic notations are also widely used. In a syllabary each character represents the sound of a syllable (for instance, Hangul for Korean, the Canadian Aboriginal Syllabics for Inuktitut, or Hiragana and Katakana for Japanese). Some scripts mix the alphabetic and phonetic mechanisms.

Characters in an ideograph-based script represent concepts. The *Hanzi* ideographs (called *Kanji* in Japanese and *Hanja* in Korean) are the most commonly known example of such a writing system. They are unified and called *Han* characters in Unicode.

Some languages use only one script; others utilize several. Japanese uses four systems. For example, the word *Nihongo* (“Japanese” in Japanese) can be written different ways, as shown in Table 1.2.

TABLE 1.2

Possible Ways to Write the Word “Japanese” in Japanese

Representation	*Script*
日本語	Kanji (ideographs)
にほんご	Hiragana (phonetic)
ニホンゴ	Katakana (phonetic, normally used to represent non-Japanese words)
Nihongo	Romaji (transliteration in Latin alphabet)

This illustrates how complex a language can be for computing.

The digital representation of these various systems leads to different ways of storing the text. The association between the different sets of characters and their computer

binary representations is called *encoding* and their implementation can be rather complex at times.

Punctuation and Marks

Punctuation signs also vary from one language to the other. In addition, some scripts use marks that have no equivalent in the Latin script.

Some languages also have additional punctuation signs. For example, Spanish uses an additional mark, an inverted question mark, at the beginning of interrogative sentences: "¿Que hora es?".

Table 1.3 lists a few examples of punctuation and marks.

TABLE 1.3

Examples of Punctuation and Marks in Various Scripts

Script	*Symbols*
Georgian	paragraph separator '჻'
Armenian	exclamation mark '՜', comma '՝', question mark '՞', full stop '։'
Arabic	comma '،', semicolon '؛', question mark '؟', percent sign '٪', ornate left parenthesis '﴾', ornate right parenthesis '﴿'
Thai	bullet '๏', ellipsis 'ฯ', section ending '๚' (*Angkhankhu*), chapter ending '๛' (*Khomut*)
Greek	question mark ';'
French	left quotation mark '«', right quotation mark '»'
Ethiopic	wordspace '፡', full stop '።', comma '፣', semicolon '፤', question mark '፧', paragraph separator '፨'
Spanish	inverted question mark '¿', inverted exclamation mark '¡'
Chinese, Japanese	comma '、', period '。', ditto mark '〃'
Tibetan	brackets '༺' (*Gug rtags gyon* mark) and '༻' (*Gug rtags gyas* mark), list enumerator '༈', topic separator '྅'

Word Separation

The use of spaces to separate words, which might look like an obvious thing to many, is not the way all languages work. Some languages, such as Amharic, use other characters to separate words.

In other cases, such as in Chinese, Korean, Japanese, and Thai, you don't separate words at all. For an example, see the following sentence in Thai:

เราจะบริหารและจัดการเว็บไซต์เป็นภาษาต่างๆเพื่อให้สามารถใช้ได้ไปทั่วโลก.

Finally, you can find languages in which spaces occur within a word. In classical Mongolian for instance, grammatical suffixes are added to the stem word separated by a narrow space, whereas another type of space separator can occur before final vowels.

As you can imagine, such properties will have a direct impact on how to handle wrapping, justification, searches, parsing, and many other aspects of text processing.

Digits

Noticeable differences between writing systems are also found in the representation of numbers.

Whereas the *Western digit* (borrowed from Arabic) is by far the most used digit system, it's not the only one. The *Arabic digits* come in two collections called *Arabic-Indic digits* and *Eastern Arabic digits* (used in Urdu and Farsi, for example). These were borrowed from India, where there are almost as many digit sets as there are scripts. In addition, Thai, Ethiopic, Khmer, Tibetan, Lao, Mongolian, and many other scripts have different digits. Most of them are listed in Table 1.4.

TABLE 1.4

Different Sets of Digits in Different Scripts

Script	*Digits*									
Western	0	1	2	3	4	5	6	7	8	9
Arabic-Indic	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Eastern Arabic	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
Bengali	০	১	২	৩	৪	৫	৬	৭	৮	৯
Devanagari	०	१	२	३	४	५	६	७	८	९
Lao	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙
Gurmukhi	੦	੧	੨	੩	੪	੫	੬	੭	੮	੯
Gujarati	૦	૧	૨	૩	૪	૫	૬	૭	૮	૯
Oriya	୦	୧	୨	୩	୪	୫	୬	୭	୮	୯
Telugu	౦	౧	౨	౩	౪	౫	౬	౭	౮	౯
Kannada	೦	೧	೨	೩	೪	೫	೬	೭	೮	೯
Malayalam	൦	൧	൨	൩	൪	൫	൬	൭	൮	൯
Tibetan	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙
Thai	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙

In addition to differences in digits, some scripts can use different numeric systems altogether. For example, Ethiopic has its own numeric system, as presented in Table 1.5.

TABLE 1.5

Ethiopic Numerals

Western	1	2	3	4	5	6	7	8	9
Ethiopic	፩	፪	፫	፬	፭	፮	፯	፰	፱
Western	10	20	30	40	50	60	70	80	90
Ethiopic	፲	፳	፴	፵	፶	፷	፸	፹	፺
Western	100	10000							
Ethiopic	፻	፼							

Chinese-based scripts usually use Western digits but also have their own set of numerals. However, in some cases accounting numbers, yet another set of numerals, are also used. Accounting numbers help to minimize the possibilities of misinterpretation or forgery. They are presented in Table 1.6.

TABLE 1.6

The Chinese Accounting Numerals

Western	0	1	2	3	4	5	6	7	8	9
Accounting	零	壹	貳	參	肆	伍	陸	柒	捌	玖
Western	10	100	1,000		10,000		100,000,000			
Accounting	拾	佰	仟		萬		億			
Western	1,000,000,000,000									
Accounting	兆									

In all cases, keep in mind that Western digits are often used, even for languages that have their own set.

Use of the correct digits is not only part of the translation of the text, it can also affect the way XML documents are rendered. For instance, the engines generating numbered lists should make provision for different types of notations.

Decimal and group separators are also different per language, and so is the way that numbers are grouped: by hundreds, by thousands, and so forth.

Writing Direction

One fundamental difference you can find in some scripts is the direction in which the text is written.

Semitic languages such as Arabic and Hebrew are displayed from right to left, top to bottom. Because they often include left-to-right runs of text or number as well, they are called *bi-directional* languages. Figure 1.3 illustrates this with a Hebrew text that includes two runs of ASCII characters.

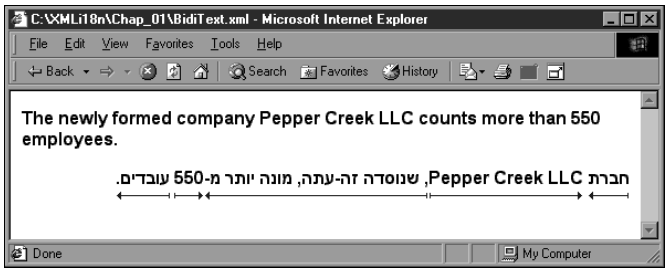

FIGURE 1.3

Bi-directional text is a mix of left-to-right and right-to-left runs of text.

Other languages, such as Urdu, Farsi, Pashto, Dhivehi, and Syriac have also adopted the Arabic writing system or scripts derived from it.

East Asian languages such as Chinese, Korean, and Japanese are traditionally written from top to bottom, with columns running from right to left. The left-to-right horizontal writing is also now commonly used.

All these variations require rendering mechanisms very different from the one used for the Roman script.

Formatting Styles

Typography choices vary from one script to another. For example, when Westerners use bold text, a Chinese or Japanese author may prefer a different method of emphasis such as dots or accent-like symbols above or below each character. The use of italics might be less appropriate in some scripts, and using uppercase to emphasize a term is impossible in non-case scripts.

Different types of layout are also used: East Asian languages can sometimes have combined characters or lines within the same row. In Arabic, justification is done with elongated connecting lines between letters (the *Kashida*), whereas some languages use wider spacing between characters and others use extra space between words.

Character Shapes

Some scripts have characteristics that are unknown or rarely used in the Latin script and offer interesting challenges. One such characteristic is multiform characters.

For instance, in Arabic, each letter does not have an upper- and lowercase form, but rather several forms depending on its location: isolated, initial, in-between, or final.

Another example is Greek, in which the letter sigma has two forms in lowercase: 'σ' and, when in terminal position, 'ς'. In uppercase both forms are converted to a unique shape: 'Σ'.

Sort Order and Case

Any presentation of a set of data must, at some point, be ordered. The mechanisms used at that time, such as collation and folding, are highly locale specific.

Languages that use ideographic-based scripts have various ways to order their characters: by phonetic, by stroke count, by radical, and so forth.

Many scripts use non–Latin-based alphabets, syllabaries, or a mixture of both. Even the languages that use the Roman alphabet such as Danish, English, and German have different ways to order it. A few examples:

- In Danish and Norwegian, 'æ' comes after 'z'', 'ø' after 'æ', and 'å' after 'ø'.
- In Swedish and Finnish, 'ü' is equivalent to 'y', 'w' is equivalent to 'v', 'å' comes after 'z', 'ä' after 'å', and 'ö' after 'ä'.
- In German 'ae', is equivalent to 'ä' that comes after 'a', 'oe' is equivalent to 'ö' that comes after 'o', 'ue' is equivalent to 'ü' that comes after 'u', and 'ß' is equivalent to 'ss'.
- In Icelandic 'ð' comes after 'd', and 'þ' after 'z'.
- In Lithuanian 'y' is equivalent to 'i'.

Related to sort order are the conventions for uppercasing and lowercasing. Although some scripts do not have such distinction, others possess very specific rules. Here are some examples:

- The German 'ß' is uppercased to 'SS'.
- The ligature 'ij' is capitalized 'IJ' in Dutch whereas it is transformed to 'Ij' in Croatian.
- In Japanese the character '〗' (U+30FC) indicates a prolonged sound for the preceding character. It changes the sorting order depending on the vowel.
- Turkish uses two letters 'i' and 'ı' that uppercase respectively to 'İ' and 'I'. This is a case in which the usual conversion rule for the ASCII characters 'i' and 'I' does not apply.

Date and Time Formatting

In addition to differences in calendars, time and dates are represented in various ways across the world: 12-hour versus 24-hour formats; the order in which month, day, and year are presented; the characters used as separators; which day is the first day of the week; and many other details make this category of information often difficult to render in a neutral fashion.

Summary

In this chapter we have drawn a brief overview of internationalization and localization. Truly integrating these two activities as part of your project development cycle is vital to ensure costs savings.

XML brings many challenges for localization: for example, lack of context when translating, processing of composite files, and the fuzzy border between source code and source document. Tools must be adapted for these issues.

A simple overview of some of the challenges brought by the use of different languages shows that although XML itself is usually well prepared for handling most languages, implementation of XML tools must follow as well.

References

The books and Web sites in the following list are some places to look for additional information about internationalization and localization methodologies, as well as the XML basics.

Nadine Kano, 1995, *Developing International Software for Windows 95 and Windows NT.* Microsoft Press. ISBN 1-55615-840-8.

Bert Esslink, 2000, *A Practical Guide to Localization*. John Benjamins B.V., ISBN 1-58811-006-0.

Apple Computer Inc., 1992, *Guide to Macintosh Software Localization*. Addison Wesley Longman, ISBN 020-1608561.

Charles Goldfarb and Paul Prescod, 2000, *The XML Handbook*. Prentice Hall, ISBN 1-850-32211-2.

Microsoft, 1993, *The GUI Guide: International Terminology for the Windows Interface*. Microsoft Press.

The XML Cover Pages: `http://www.oasis-open.org/cover/xml.html`

XML architecture: `http://www.w3.org/XML`