Introduction 1

What Is Internationalization?

Internationalization is a way of designing and producing software that can easily be adapted to local markets. Internationalized products can be **localized** or adapted to different languages and cultures with minimal effort.

How Is Localization Different?

Internationalization is the process of making software portable between languages or regions, while localization is the process of adapting software for specific languages or regions. International software can be developed using interfaces that modify program behavior at run time in accordance with specific cultural requirements. Localization involves establishing on-line information to support a language or region, called a *locale*.

Unlike software that must be completely rewritten before it can work with different native languages and customs, internationalized software does not require rewriting. It can be ported from one locale to another without change. The Solaris system is internationalized, providing the infrastructure and interfaces you need to create internationalized software. Chapter 3, "Support for Internationalization" and Chapter 4, "Writing Internationalized Code" describe what facilities are available and how to use them.

The `localedef`(1) utility simplifies the process of generating a locale.

1

Advantages of Internationalization

Creating internationalized software automatically expands the market for your product. If you follow the steps outlined in this book, your product will be compatible with all the languages and cultures supported by Solaris.

Many computer firms, including Sun Microsystems® and Digital Equipment, obtain around half of their revenue from outside the United States. Profit margins are often higher abroad, so contribution to net income can be even higher. Localization costs can be high, particularly for translation, but often repay themselves quickly in higher sales.

Basic Steps in Internationalization

An internationalized application's executable image is portable between languages and regions. To internationalize software, you:

- Use the interfaces described in this book to create software whose environment can be modified dynamically without the software needing to be recompiled.
- Separate all printable and displayable messages that the user sees from the executable image. Keep these message strings in a message database.

Message strings are translated for a language and region (called a locale) as part of the localization process. Related databases that specify formats for time, currency, and numbers are translated at the same time.

To use a localized version of a product, users set an environment variable. The product then displays user messages in their translated form, and also formats date, time, and currency according to the locale-specific conventions. Thus, users gain control of their software's language and behavior.

Conforming to Standards

Many standards bodies are developing guidelines for internationalized software. Practices described in this book conform to various ANSI, IEEE, ISO, and X/Open™ standards. International standards are still evolving, so not all interfaces can be guaranteed forever.

Internationalization Levels

SunSoft™ defines four levels of internationalization, described below. Levels are not necessarily hierarchical, but do indicate difficulty of implementation.

Level 1—Text and Codesets

Software that is level-1 compliant is "8-bit clean" and can therefore use the ISO 8859-1 (also called ISO Latin-1) codeset. Historically, many programmers assumed their application needed only the ASCII character set. Because the ASCII codeset employs only seven bits out of an 8-bit byte, the most significant bit was often used to store information about the character. For example, setting the most significant bit "on" might indicate that the character is highlighted. The ISO Latin-1 codeset employs all eight bits. Software that uses the most significant bit for its own purposes is not level-1 compliant.

Level 2—Formats and Collation

Software is level-2 compliant if its formatting and collation methods are locale sensitive. Many different formats are employed throughout the world to represent date, time, currency, numbers, and units. Also, some alphabets have more letters than others, and the order in which letters are sorted within national alphabets varies from language to language. Programs that leave the format design and sorting order to the localization center in a particular country are considered level-2 compliant.

Level 3—Messages and Text Presentation

User-visible text in a level-3 compliant application should be easily translatable into the languages of various target markets. User-visible text includes help text, error messages, property sheets, buttons, text on icons, and so forth. A common way to provide message translation is to put text to be translated into a separate file with messages indexed either by string contents (the POSIX/UniForum method) or by number (the XPG-3 method). Software using separate message files for messaging—rather than encapsulating user messages in the binary—is considered level-3 compliant.

The Solaris system provides two different but incompatible methods of translating text. The first and preferred method is the POSIX and UniForum proposed standard using the `gettext()` function.

The second method of message translating is XPG-3 style *message catalogs*. XPG is the X/Open Portability Guide. XPG-3 provides the `catgets()` function to obtain translated message strings from a message database. The XPG message translation standard is not recommended and is not used in Solaris.

Whichever mechanism is used for messages translation—and only one method should be used in any one application—level-3 compliant software should strive to avoid compound messages (messages consisting of separately composed parts) because word order is different in many languages.

Level 4—Asian Language Support

Level-4 compliant software provides support for East Asian languages, which often require multibyte codesets because of their large character inventory. The Solaris system provides such support with the Extended Unix Code (EUC). This is a method for switching between multiple codesets, three of which may in turn be multibyte codesets.