THE X/OPEN INTERNATIONALIZATION MODEL

By Wendy Rannenberg and Jürgen Bettels

ABSTRACT

Software internationalization standards allow developers to create applications that are neutral with respect to language and cultural information. X/Open adopted a model for internationalization and has revised the model several times to expand the range of support. The latest version of the X/Open internationalization model, which supports multibyte code sets, provides a set of interfaces that enables users in most of Europe and Asia to develop portable applications independent of the language and code set. One implementation of this model, the internationalized DEC OSF/1 AXP version 1.2 (based on OSF/1 release 1.2) supports complex Asian languages such as Chinese and Japanese.

INTRODUCTION

Software internationalization standards initiatives began in the late 1980s. This paper provides a brief history of internationalization standards activities followed by a description and an analysis of the X/Open model for internationalization. The Open Software Foundation's OSF/1 release 1.2 and Digital's DEC OSF/1 AXP version 1.2 internationalization implementations serve as reference software for the description. The analysis covers both the strengths and the limitations of the model. The paper concludes with a discussion of current and future relationships between this model and other work in the field.

INTERNATIONALIZATION STANDARDS

The International Organization for Standardization (ISO) is the primary group that is currently publishing or developing internationalization specifications, including code sets, programming languages, and frameworks. Before the ISO adopts emerging specifications, much work is done by other groups. In the case of interfaces that support the development of international applications, the Uniforum Internationalization Technical Work Group, the X/Open Internationalization Work Group, the Unicode Consortium, and the X Consortium have been instrumental.

Internationalization is generally considered to be the processes and tools applied to create software that is neutral with respect to language and cultural information. This neutrality can be accomplished by providing a set of application interfaces designed to isolate sensitivity to language and culture-specific information. Such interfaces include functionality to

- Attain character attributes independent of coded character sets, i.e., code sets
- Order relationships of characters and strings
- Process culturally sensitive format conversion (e.g., date, time, and numbers)
- Maintain user messages for multiple languages

Standardization of internationalization interfaces began predominantly in the UNIX environment. Companies such as Hewlett-Packard and AT&T provided early proprietary solutions.[1]

When X/Open announced its intention to include support for internationalization in Issue 2 of its X/Open Portability Guide (XPG2), Hewlett-Packard submitted its Natural Language Support System as a proposal for an internationalization model. X/Open further developed this proposal and published the guide in 1987.[2] Some principles developed for these solutions found their way into the emerging C programming language standard (ISO/IEC 9899) and the POSIX operating system interface specification (ISO/IEC 9945-1).[3,4]

The subsequent version of the X/Open Portability Guide, XPG3, published in 1989, demonstrated further improvement in internationalization support.[5] The guide was aligned with the ISO/IEC C standard and the ISO/IEC POSIX specification, both of which meanwhile had been finalized.

A major drawback of the XPG3 specification is that it is limited to single-byte code sets. Such code sets are used primarily for western European languages and preclude use of the X/Open internationalization model for Asian and eastern European languages.

The Japanese UNIX Advisory Group developed specifications to extend support to character sets that are encoded in more than one byte. These code sets are generally known as multibyte code sets. The Multibyte Support Extensions developed by this group are now included in an addendum to the ISO/IEC C programming language standard.[6] This work was also adopted by X/Open for inclusion in Issue 4 of the X/Open Portability Guide (XPG4), which was published in 1992.[7,8,9]

However, the underlying model used by X/Open and POSIX does not fully meet the needs of distributed and multilingual computing environments. Therefore, in 1992 X/Open and Uniforum created a joint internationalization work group, commonly referred to as the XoJIG. This group investigated internationalization requirements for distributed and multilingual environments and, in November 1992, published a revised model for internationalization.[10]

THE X/OPEN INTERNATIONALIZATION MODEL

When X/Open first investigated the need for internationalization services, several needs were identified:

- Meet the market requirements of the X/Open member companies. (Many of these requirements were based on the needs of the European Economic Community [EEC].)
- Support more than one language and cultural environment, including messages and date/time.
- Provide for data transparency, i.e., remove 7-bit, U.S. ASCII restrictions from the environment.

As discussed previously, X/Open adopted a model for internationalization and has updated and revised the model many times. The next section describes the current X/Open model.

Overview of the X/Open Portability Model, Issue 4

There are five components to the current X/Open internationalization model, X/Open Portability Guide, Issue 4 (XPG4):

1. Locale announcement mechanism
2. Locale databases
3. Internationalization-specific library routines
4. Internationalized interface definitions for standard C language library routines
5. Message catalog subsystem

The locale announcement mechanism provides a way for an application to load, at run time, a specific set of data that describes a user's native language and cultural information. An application user can specify a language, a territory, and a code set by means of environment variables. The locale announcement mechanism checks the environment variables. If the variables are set, the application attempts to load the locale-specific data. If the environment variables are not set, most applications default to the use of the POSIX (i.e., C language) locale or an implementation-defined locale. The POSIX locale definition is based on the U.S. ASCII code set and the U.S. English language.

In conjunction with locale databases, the announcement mechanism provides access to code set specification data, character collation information, date/time/numerical/monetary formatting

information, negative/affirmative responses, and application-specific message catalogs.

Figure 1 shows the relationships among the components of the X/Open internationalization model.[11] Refer to Figure 1 throughout this section, as the various elements of the figure are described.

[Figure 1 (Components of the X/Open Internationalization Model) is not available in ASCII format.]

The locale announcement mechanism is based on the setlocale() function

```
char *setlocale(int category,
  const char *locale)
```

The categories correspond to components of the locale database and have a set of corresponding user environment variables. The announcement mechanism supports an order of precedence when querying the user's environment to establish the preferred locale. Table 1 shows the environment variables specified by XPG4.

Table 1 Locale-specific Environment Variables

Variable	Use
LC_ALL	For all categories
LC_COLLATE	For collation
LC_CTYPE	For character classification
LC_MESSAGES	For responses and message catalogs
LC_MONETARY	For monetary information
LC_NUMERIC	For numeric information
LC_TIME	For date/time information
LANG	If no others are set

If no others are set The LC_ALL environment variable has precedence over all others, whereas the LANG environment variable has no precedence. The other LC_* environment variables are of equal weight.

Although it does not provide a naming convention for locales, the X/Open model does specify the locale argument as a pointer to a string in the form

XPG3:
language[_territory[.codeset]][@modifier]

XPG4:
language[_territory][.codeset][@modifier]

Examples of environment variable settings are

```
  LANG=en_US.ISO8859-1
```

and

```
  LC_COLLATE=ja_JP.jpEUC
```

The modifier is sometimes used to specify a particular instance of a language or cultural information for a locale. For instance, if support for a particular sort order is necessary, in a German locale the user might specify

```
 LC_COLLATE=de_DE.ISO8859-1@phone
```

to sort alphabetically according to the telephone directory rather than the dictionary.

Locale databases can be provided by either the system vendor or an application developer. A description of utilities that convert a source format specification of a locale to a binary file follows.

The setlocale() function accesses the binary locale databases and provides a global locale within a given application. The global locale is similar to a global variable in that it is shared by all of an application's procedures. Locale switching can be done within an application, but within the scope of the XPG4 model such locale switching is unnecessarily complex and costly, in terms of performance. A later section discusses additional limitations of this mechanism.

The set of interfaces shown in Table 2 supports international application development and was first introduced as part of the ISO/IEC C and the XPG2 and XPG3 specifications. These interfaces are used primarily to access data in the locale databases or to manipulate locale-sensitive data.

Table 2 Interfaces for International Application Development

Interface	Use
localeconv()	For retrieving locale-dependent formatting parameters
nl_langinfo()	For extracting information from the locale database
setlocale()	For locale announcement
strcoll()	For locale-based string collation
strftime()	For converting date/time formats based on locale
strxfrm()	For transforming a string for collation in current locale

The XPG3 specification is based on the use of ISO/IEC 8859-1 as the transmission code set.[12] Some implementations use this as an internal code set, instead of the ASCII code set.

A limited set of functions that support multibyte characters is also available: mblen(), mbtowc(), mbtowcs(), wctomb(), and

wcstombs(). Each of these functions is based on the ISO/IEC C wide character (wchar_t) data type. The size of the data type is not specified by the standard and can vary from one implementation to the next, depending on the code set support offered by a particular vendor. This multibyte function set does not provide adequate support for Asian language application development.

In addition to the mb* and wc* functions, the X/Open internationalization model specifies a set of extensions for many library functions and commands. These extensions enable the support of 8-bit characters as well as provide the functionality required to meet the original goal of ensuring data transparency. For example, changes to the printf() and scanf() families of functions allow the ordering of arguments to be specified in translated message catalogs. In addition, about 80 commands, including sort and date, were modified to support the locale categories.

The XPG specifications include a message catalog subsystem. Although not very sophisticated, this subsystem provides much needed functionality. Minor updates have been made with each new issue of the Portability Guide. The subsystem comprises only three functions: catopen(), catclose(), and catgets(). A command, gencat, is used to convert a message source file into a binary message catalog that is accessed at run time by an application. The behavior of the catopen() function is dependent on the user's chosen locale allowing selection of translated messages.

XPG4 Specification and the OSF/1 Release 1.2 Implementation

This section discusses the XPG4 model in terms of the OSF/1 release 1.2 implementation. Topics include code set support, the locale definition utility (the utility for handling data in mixed code sets), worldwide portability interfaces, and local language support.

Code Set Support. As mentioned in the previous section, the XPG3 specification primarily supports code sets based on the ISO/IEC 8859-1 specification. The XPG4 model goes beyond this by including additional interface specifications to support multibyte locales and internationalized commands.

The XPG4 model is a superset of the five basic components of the XPG3 model. The use of the wchar_t data type is a key feature of the new interface specifications, because this data type supports multibyte code sets. In the internationalized DEC OSF/1 AXP version 1.2 system, the size of wchar_t is 32 bits, which enables the support of complex Asian languages such as Chinese. This implementation is based on the OSF/1 release 1.2, which is itself designed to support 8-, 16-, or 32-bit wchar_t definitions. The X/Open internationalization model is based on the concept of

process and file codes. In the internationalized DEC OSF/1 version 1.2 implementation, the wchar_t data type is used as process code. That is, internal to an application, characters are converted to the wchar_t data type before use. File code, i.e., on-disk data, is always stored as multibyte characters. An application converts all internal process code (i.e., wchar_t data type) character to multibyte character prior to storing it on disk. This enables file compression and enforces the use of a constant width for the processing of character information. The mb* and wc* functions convert between the two types of data. The size of the wchar_t data type combined with the capability to support multiple encoding schemes provides the flexibility required to have a code set-independent implementation.

Restrictions exist on the use of certain characters in the second and subsequent bytes of a multibyte character so that full code set independence is difficult to achieve. An example of such a restriction is the slash character /. The UNIX file system uses this character as a delimiter in absolute and relative pathname specifications. Implementations based on OSF/1 release 1.2 restrict the use of characters in the range 0x00-0x3F to the ASCII code set. However, even with this restriction, it is possible to build robust systems that support a wide range of multibyte code sets.

To gain the necessary flexibility, the Open Software Foundation introduced an object-oriented architecture for the internationalization subsystem. This architecture specifies the various components of the X/Open model as subclasses. At run time, an application instantiates objects built from these subclasses by means of the setlocale() function call.

localedef, iconv and Code Set Independence. XPG3 does not provide a utility for describing locales. Therefore, the number of different approaches to the problem matched the number of vendors. Introduced in the POSIX specification ISO/IEC DIS 9945-2 and hence adopted by X/Open, the localedef utility provides a mechanism for specifying a locale in a portable manner.[13] For each code set supported in the internationalized DEC OSF/1 AXP system, there is a corresponding charmap file and one or more corresponding locale definition files that adhere to the POSIX specifications. Combined with a set of locale-specific methods and code set converter modules, these subclasses provide the foundation for the OSF internationalization architecture.

Locale-specific methods provide a way for the ISO/IEC C language mbtowc(), wctomb() family of functions to work in a multiple code set environment. The wchar_t encoding of a multibyte character in the Japanese SJIS code set is different from that for a character in the Super DEC Kanji code set. At execution time, the correct method is instantiated based on the user's choice of locale. An example of such an instantiation is shown in Figure 2.

Figure 2 Instantiation of mbtowc()

```
  LANG=ja_JP.SJIS

    mbtowc()  -- > sjis_mbtowc()

  or

  LANG=zh_TW.eucTW

    mbtowc()  -- > eucTW_mbtowc()
```

A user-level utility (iconv) and several library functions (iconv(), iconv_open(), and iconv_close()) provide a way to handle data that may be in mixed code sets. Internationalized DEC OSF/1 version 1.2 provides an extensive set of code set conversion modules. New conversion methods are easily added to the system.

Worldwide Portability Interfaces. The XPG4 internationalization architecture parallels the XPG3/ISO C model. For example, XPG4 specifies a family of isw* functions similar in design to the is* functions (e.g., isalpha) specified in the ISO/IEC C standard. As mentioned previously, the XPG3 model does not include all the interfaces necessary for application developers to handle multibyte code sets. A new set of interfaces, which parallels the set of ISO/IEC C 8-bit interfaces, was developed and integrated into the XPG4 specification. The final version of the interface specification was proposed to the ISO/IEC C committee as the Multibyte Support Extensions.

Cultural Data/Local Language Support. Local language support is achieved through the use of locale databases and message catalogs. The catalogs enable translation of user messages. Locale databases have two components: the charmap file and the locale definition file. These databases are created by means of the localedef command.

The charmap file contains a POSIX-compliant specification of the code set, i.e., a one-to-one mapping from character to code point. The locale definition file contains the cultural information. Various sections of the definition file correspond to the categories referenced by the setlocale() function. The definition file contains collation specifications, numeric and monetary formatting information, date/time formats, affirmative/negative response specifications, and character classification information. In the OSF/1 release 1.2 implementation, these definition files are independent of the code set. For example, the definition for Japanese (ja_JP) can be combined with multiple charmap files such as SJIS or eucJP.

STRENGTHS OF THE X/OPEN MODEL

The greatest strength of the X/Open internationalization model is that it is in place today and enables the development of portable, language- and code set-independent applications. The internationalized DEC OSF/1 AXP version 1.2 system provides support throughout the commands and utilities for 20 code sets that represent major European and Asian languages. All this is accomplished using XPG4 application programming interfaces (APIs). In addition, the programming paradigm is consistent with ANSI C, making it easier for application developers to modify existing applications for international support.

LIMITATIONS OF THE X/OPEN MODEL

As described previously, the X/Open model for internationalization provides a comprehensive set of application interfaces, thus enabling the development of applications that can be used worldwide. Yet, as with many standards, there are limits to what can be accomplished. In this case, limitations manifest themselves in several areas:

- C language API
- Distributed computing environments
- Multithreaded applications
- Multilingual applications[14]
- Unicode and ISO/IEC 10646 support[15,16]

Because the X/Open and POSIX specifications are based on UNIX implementations, the APIs are specified only for the C programming language. For programming languages such as COBOL, FORTRAN, and Ada, it is not necessarily possible to match the syntax and semantics of the API. The remainder of this section explores generic problems with the global locale model and addresses specific issues in more detail.

Global Locale Issues

The X/Open model is based on the concept of a global locale. This aspect of the model is achieved through the use of locale data that is maintained in a private, process-wide global structure. The use of a global locale is one of the more severe drawbacks to using the overall model.

When working with this model, application developers typically assume that a single language-territory-code set combination is in use at a given time and will remain constant on a per-process basis. Although it is possible to use the announcement mechanism

to determine the run-time locale of a process, this mechanism is cumbersome. The application must both save and restore the locale information.

Another drawback of the X/Open model is that existing APIs do not include a way to share locale-specific information between processes. This, combined with the difficulty of locale switching, limits the ability to support multilingual and distributed applications.

Distributed Processing Issues

In a client-server environment, the problem of supporting multiple locales becomes a serious issue. Consider the following examples:

- A server gets requests from various clients, each running their own locale. These requests are processed using the locale of the client. The process includes returning locale-specific user messages to the client and processing user-locale-sensitive date/time formats, collation information, and string manipulation.

- A window manager that supports multiple clients displays menus for a client based on the client's locale. The user error messages displayed are based on the locale of the server.

When a client sends a request to a server, the request parameters that are passed between the client and the server imply an associated locale. Since the global locale is not an explicit argument in any of the XPG4 functions, this locale is difficult to pass to the server. Consider the specific case of remote procedure calls (RPCs), where an interface definition language (IDL) might be used to generate client stubs. Because of the global nature of the locale, insufficient information is available to the IDL to determine if the locale information needs to be used as an argument to any generated functions. Thus, the server may need to change its locale for each client request, which may be unacceptable in terms of system performance.

Using the current XPG model, synchronizing the use of a specific locale between a client and server may not be possible. Even if a client could specify a locale as part of the request, the locale may not be available at the server side or may be replicated incorrectly on the server side. This situation exists because locale names and content are not standardized.

Although the XPG4 specification includes the localedef command for specifying the content of a locale database, there is no provision for standardizing the content. The only locale for which an X/Open specification exists is the POSIX or C locale. In

addition, there is no specification for explicitly naming a locale. Locale names are composed of language, territory, and code set components. Many vendors use ISO/IEC 639 and ISO/IEC 3166 for the language and territory components, but there is little agreement on code set naming conventions.[17,18] This naming scheme is not sufficient for uniquely identifying locales, as is required in a client-server model.

Another problem with the X/Open model that impacts application performance and the ease with which an application can be internationalized is related to the process code. The representation of the process code, i.e., wchar_t, is implementation defined, and the mapping of multibyte characters to wide character codes may be locale sensitive. Therefore, wchar_t-encoded data cannot be exchanged freely between the client-server pair. The only exception would be if the end user guaranteed that the process code was identical for a given locale for each part of the client-server pair. The XPG4 specification does not include functionality to identify or to interrogate the wchar_t encoding scheme used.

Multithreaded Applications

The problems encountered in a distributed processing environment become more complex if the application is also multithreaded. Using POSIX threads, commonly referred to as pthreads, more than one thread is in the execution phase at the same time.[19] Again, a problem with the global, process-wide locale is evident. The application cannot maintain the state of the global locale, accomplished by a save/restore process, without blocking all other threads. Likewise, execution of locale-sensitive functions requires locking all threads to ensure that the global state is not altered prior to completion. The need to continually lock and unlock threads, in addition to being undesirable, results in a performance problem for internationalized applications. Another approach is to make locale data thread-specific.

Multilingual Applications

The X/Open internationalization model is oriented toward the development of monolingual applications. Therefore, the model does not provide functions to handle data that consists of an arbitrary mixture of languages and code sets.

The following are some examples of applications that may require multilingual services:

- Applications that simultaneously interact with a number of users (e.g., transaction processing systems), where each user can choose a language

- A word processing application for multilingual texts that

need language-sensitive formatting, hyphenation, etc.

Unicode Support

With the arrival of the Unicode universal character code and the adoption of ISO/IEC 10646 as its form, both POSIX and X/Open have to address the issues of support.[15,16] The X/Open Internationalization Working Group is preparing a paper on Unicode support within the existing specifications; this publication should be available in late 1993. Some of the issues that the C language, POSIX, and XPG4 are facing to support Unicode or ISO/IEC 10646 are character compatibility, code restrictions, and valid character strings.

Unicode characters are incompatible with the C language char* data type used in the POSIX and X/Open models. Unicode characters are 16-bit entities, whereas the POSIX and X/Open characters are in practice 8-bit bytes, even though theoretically the byte size is implementation dependent. Most APIs defined in the POSIX and X/Open models implicitly assume 8-bit characters. This principle is extended to cover Asian multibyte characters by considering each character to be a sequence of 8-bit char data elements. Unicode characters, however, cannot be broken down into sequences of valid 8-bit char* data elements.

The POSIX character model requires that the code values for char* data protect the code ranges for control characters between 0x00-0x1F and 0x80-0x9F, the code position DELETE, and the slash character /. No such restrictions exist in Unicode.

The C language postulates that a null character terminates a char* string. Since the Unicode string most likely contains zero bytes, these bytes would be interpreted as string terminators. In principle, the C language would allow a compiler to define the char* data type to be of 16-bit width. However, given the prevailing assumption in POSIX and XPG4 that one character equals one 8-bit byte, a Unicode character string cannot be a valid char* string.

For these reasons, Unicode cannot be a valid file code as defined by the POSIX and X/Open specifications. Unicode is not usable as an XPG4 process code either. Unicode and ISO/IEC 10646 allow the combining of 16-bit characters.[15] However, in many operations the combining character (e.g., in the French character set, the grave accent) and the base character (e.g., the letter e) have to be processed together. This situation contradicts the XPG4 model, where each character of the process code is individually addressed and processed.

Using a well-defined encoding as XPG4 process code would also violate the principle that the process code is opaque, implementation defined, and not valid outside the current process. For all these reasons, the X/Open Joint

Internationalization Group decided to propose using Unicode in a modified form of the universal multiple-octet coded character set (UCS) transformation format (UTF).[16,20]

PROPOSED CHANGES TO THE MODEL

The XPG4 model limitations described in the previous sections are well understood in the internationalization community. X/Open has published a Snapshot specification for a set of distributed internationalization services.[10] This specification does not solve all the problems identified in this paper. It does, however, address the problems associated with the use of the global locale mechanism, locale identification, and text object manipulation. Note that these are proposed changes and have not been adopted by any standards organization.

The proposed changes include

- A locale naming specification that enables the identification of a given locale in a distributed environment
- Definition and support of a locale registry
- A new set of APIs that enables application software to
  - Concurrently manage and use many different locales
  - Manipulate opaque text objects[21]
  - Support stateful and nonstateful encodings and file codes that are excluded by the current standards (e.g., nonzero byte terminators used in the Unicode code set)

Locale Naming and the Locale Registry

In an internationalized environment, the server must replicate the client's locale. If the client's locale can be uniquely identified, the remote code can replicate the locale by obtaining it and specifying this information as part of the operation. To solve the locale replication problem, the XoJIG developed a locale naming scheme, referred to as the locale specification.

The locale specification is a character string that contains the locale name for each category that exists within the locale. The syntax for locale names is a list of keyword-value pairs, where each pair defines a locale category. Certain keywords, such as code set name, encoding name, and owner or vendor name, are standardized as part of the registration process. Table 3 shows two examples of locale specifications.

Table 3 Network Locale Naming Specifications

American English Locale Using the ISO/IEC Latin-1 Code Set

```
CTYPE=ANSI;en_US;01_00;ISO-88591-1987;;/
COLLATE=ANSI;en_US;01_00;ISO-88591-1987;;/
MESSAGES=ANSI;en_US;01_00;ISO-88591-1987;;/
MONETARY=ANSI;en_US;01_00;ISO-88591-1987;;/
NUMERIC=ANSI;en_US;01_00;ISO-88591-1987;;/
TIME=ANSI;en_US;01_00;ISO-88591-1987;;/
```

Japanese Locale Using Japanese Extended UNIX Code (EUC) Encoding

```
CTYPE=ISO;ja_JP;01_00;JIS-X0208-1987,JIS-X0201-1987,JIS-X0212-1991;EUC;/
COLLATE=ISO;ja_JP;01_00;JIS-X0208-1987,JIS-X0201-1987,JIS-X0212-1991;EUC;/
MESSAGES=ISO;ja_JP;01_00;JIS-X0208-1987,JIS-X0201-1987,JIS-X0212-1991;EUC;/
MONETARY=ISO;ja_JP;01_00;JIS-X0208-1987,JIS-X0201-1987,JIS-X0212-1991;EUC;/
NUMERIC=ISO;ja_JP;01_00;JIS-X0208-1987,JIS-X0201-1987,JIS-X0212-1991;EUC;/
TIME=ISO;ja_JP;01_00;JIS-X0208-1987,JIS-X0201-1987,JIS-X0212-1991;EUC;/
```

Although this naming scheme provides for unique identification of locales, the names are long. The specification calls for the use of ASCII characters to name locales. The American English locale specification is over 200 bytes in length. A shorthand notation called network locale specification token has been proposed.

The network locale specification token is an unsigned integer value that can be represented within four bytes. The two most significant bytes represent the registration authority. Under the proposal, national and international standards bodies, companies, and consortia, etc., that wish to use network locale specification tokens will receive unique identifiers. A block of values will be reserved for private use between consenting systems. A set of new functions will allow conversion between the full locale specification and the locale specification token.

The locale specification proposal solves the problem of unique naming for locales. Combined with a locale registry, this proposal overcomes some of the limitations of the current X/Open model. Within the registry, each locale will have a name defined according to the new syntax. Assuming vendors add these registered locales to their systems, language-sensitive operations in a distributed environment will obtain the same results across systems. This registry has been established by X/Open, and several locales have been submitted.

Multilocale Support

A new set of interfaces, the set of o* functions, has been proposed. These interfaces provide capabilities similar to those defined by the XPG4 model. These new functions address many of the model's limitations, including multithreaded applications,

distributed systems, and multilingual applications.

Most of the o* functions utilize three new data types: locale object, attribute object, and text object. To overcome the limitation imposed by a global, per-process locale, the fundamental XPG4 programming paradigm is altered to define localization on a per-call rather than a per-process basis. This change is accomplished by defining a new opaque data type called a locale object. A locale object identifies the locale and can be passed as an argument to locale-sensitive functions on a per-call basis. In this way, the basic programming paradigm becomes

1. Perform operation X on data Y using locale Z

and not

1. Set global locale Z
2. Perform operation X on data Y

An attribute object is a generic opaque object that serves as a container to other opaque objects, such as a locale object. Use of an attribute object in the proposed APIs provides a solution that is not specific to solving internationalization problems. It is anticipated that objects, in addition to the locale object, will be identified. The additional objects might result from requirements in such areas as multimedia, network security, and X11-specific extensions to the locale.

A text object is a new data type that replaces the character (char) and wide character (wchar_t) data types used in the XPG4 internationalization model. As previously defined, a text object refers to a collection of text characters that may or may not have metadata associated with them. Support for directionality, as required for right-to-left languages such as Hebrew, is an example of when such metadata would be introduced. If a text object has a locale defined as part of the metadata (i.e., self-announcing data), the locale specified as part of the data supersedes the locale passed as an argument to the o* functions. The locale that is passed as a function argument acts as a default locale for operations that require it. All o* functions allow a locale identifier to be passed as an argument. This capability eliminates the limitations of the XPG4 global locale. The support of metadata associated with text objects is implementation defined.

A text object data type is represented by a text pointer of type txt_ptr. A text pointer represents all the information associated with a particular character position within the text object. This information is sufficient to perform any kind of operation, such as classification, extraction, or uppercasing.

In summary, the o* functions allow text objects to be classified,

converted, transferred to and from files, etc. The functionality of the o* functions is designed to parallel the character-handling functionality provided by the X/Open internationalization model. For example, functions for manipulating text pointers and for concatenating text objects are tuned to the multilocale model. Interfaces have also been introduced to provide management functions for new objects.

CONCLUSIONS

When introduced, the X/Open Portability Guide Issue 3 model for internationalization met about 90 percent of the known requirements in the western European market. The introduction of the XPG4 worldwide portability interfaces expanded the region to include Asia, Japan, and eastern Europe. Consequently, application developers can write portable code that supports a variety of languages. The use of the worldwide portability interfaces for computer-aided design applications that are distributed worldwide is one example of such code.

However, the use of the client-server model expanded greatly in the time it took to develop these standards. Also, the need to support truly multilingual applications in a distributed environment became evident. New code set specifications (i.e., Unicode) have been adopted, and systems supporting Unicode as both file and process code have been implemented. Application vendors are beginning to see their markets expand into every corner of the world.

The XPG4 model will continue to provide much-needed interfaces for quite some time. Yet, to meet the challenges of the truly distributed environment, a new API, similar to the o* functions presented here, must be developed and accepted.

ACKNOWLEDGMENTS

Thanks to Mike Feldman, Richard Hart, and Dave Lindner, among others, who spent their time providing comments and recommendations during the writing of this paper.

REFERENCES AND NOTES

1. UNIX System V Release 4 Multi-National Language Supplement (SVR4 MNLS) Product Overview (Japan: American Telephone and Telegraph Co., 1990).

2. X/Open Portability Guide, Issue 2 (Reading, U.K.: X/Open Company Ltd., 1987).

3. Programming Languages -- C, ISO/IEC 9899:1990 (Geneva: International Organization for Standardization/International

Electrotechnical Commission, 1990).

4. Information Technology -- Portable Operating System Interface (POSIX) -- Part 1: System Application Program Interface (API) [C Language], ISO/IEC 9945-1:1990 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1990).

5. X/Open Portability Guide, Issue 3 (Reading, U.K.: X/Open Company Ltd., 1989).

6. Multibyte Support Extensions, ISO/IEC 9899:1990/Amendment 3:1993(E) (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1993).

7. X/Open CAE Specification, System Interface Definitions, Issue 4, ISBN 1-872630-46-4, C204 (Reading, U.K.: X/Open Company Ltd., 1992).

8. X/Open CAE Specification, Commands and Utilities, Issue 4, ISBN 1-872630-48-0, C203 (Reading, U.K.: X/Open Company Ltd., 1992).

9. X/Open Internationalisation Guide (Reading, U.K.: X/Open Company Ltd., 1992).

10. Distributed Internationalization Services (Snapshot) (Reading, U.K.: X/Open Company Ltd., 1992).

11. L. Laverdure, P. Srite, and J. Colonna-Romano, NAS Architecture Reference Manual (Maynard, MA: Digital Press, 1993): 255-264.

12. Information Processing -- 8-bit, Single-byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1, ISO/IEC 8859-1 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1987).

13. Information Technology -- Portable Operating System Interface (POSIX) -- Shell and Utilities, ISO/IEC DIS 9945-2 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1992).

14. Multilingual applications can process multiple languages at the same time, whereas implementations of the X/Open model can process several languages but only on an individual basis.

15. J. Bettels and F. Bishop, "Unicode: A Universal Character Code," Digital Technical Journal, vol. 5, no. 3 (Summer 1993): 21-31.

16. Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1993).

17. Codes for the Representation of Names and Languages, ISO/IEC 639 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1988).

18. Codes for the Representation of Names of Countries, ISO/IEC 3166 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1988).

19. Information Technology -- Portable Operating System Interface (POSIX) -- Threads Extension for Portable Operating Systems, IEEE 1003.4a/D7 (New York, NY: The Institute of Electrical and Electronics Engineers, 1993).

20. File System Safe -- UCS Transformation Format (Reading, U.K.: X/Open Company Ltd., 1993).

21. As defined in the X/Open Draft Internationalization Services Snapshot: A text object is an implementation-defined representation of a fragment of text that consists of zero or more text characters.

GENERAL REFERENCE

S. Martin and M. Mori, Internationalization in OSF/1 Release 1.1 (Cambridge, MA: Open Software Foundation, Inc., 1992).

TRADEMARKS

BIOGRAPHIES

Jürgen Bettels Jürgen Bettels is an internationalization architect and the standards manager for the International Systems Engineering Group. Since 1986, he has worked on a number of internationalization architectures starting with DECwindows. He participated in the Unicode consortium, ECMA, and X/Open on internationalization. He contributed to the ISO/IEC WG2/SC2, whose work merged Unicode and ISO 10646 into a single universal character encoding. Prior to joining Digital, he was a physicist at the European particle laboratory, CERN, in Geneva. Jürgen has the degree of Diplom Physiker (physicist) from the University of Aachen.

Wendy Rannenberg Principal software engineer Wendy Rannenberg manages the UNIX Software Group's internationalization team. She is responsible for the delivery of Digital's internationalization technology on both the ULTRIX and the DEC OSF/1 AXP platforms. Prior to joining Digital in 1988, she held engineering positions with Lockheed Sanders Associates and the Naval Underwater Systems Center. Wendy holds a B.S. (1980) in engineering from the University of Connecticut at Storrs and is a member of IEEE, SWE, and ACM. She has written or contributed to numerous technical publications.