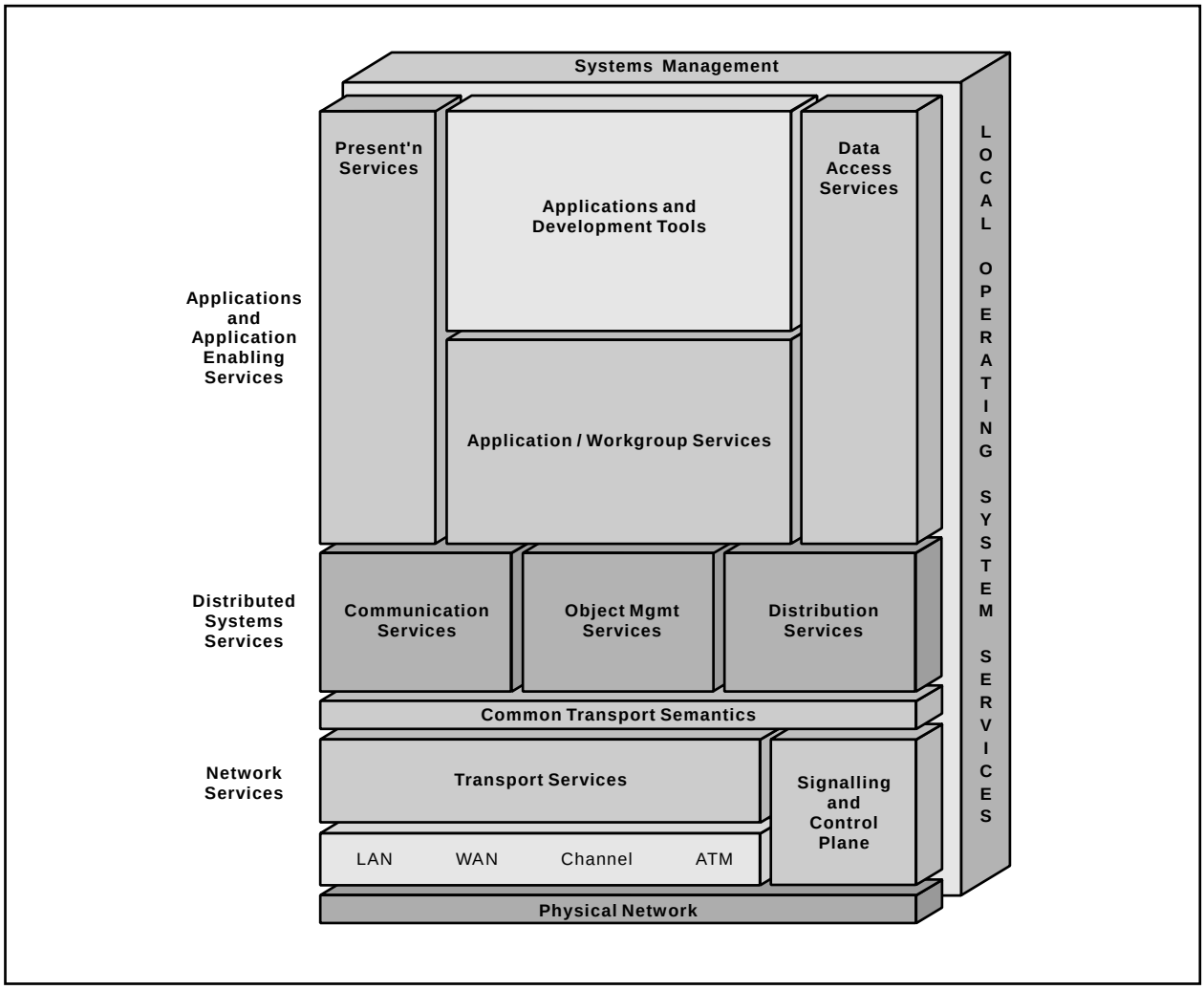
Open Blueprint

Internationalization Resource Manager



G325-6583-00

Open Blueprint

Internationalization Resource Manager

G325-6583-00

About This Paper

Open, distributed computing of all forms, including client/server and network computing, is the model that is driving the rapid evolution of information technology today. The Open Blueprint structure is IBM's industry-leading architectural framework for distributed computing in a multivendor, heterogeneous environment. This paper describes the Internationalization resource manager component of the Open Blueprint and its relationships with other Open Blueprint components.

The Open Blueprint structure continues to accommodate advances in technology and incorporate emerging standards and protocols as information technology needs and capabilities evolve. For example, the structure now incorporates digital library, object-oriented and mobile technologies, and support for internet-enabled applications. Thus, this document is a snapshot at a particular point in time. The Open Blueprint structure will continue to evolve as new technologies emerge.

This paper is one in a series of papers available in the *Open Blueprint Technical Reference Library* collection, SBOF-8702 (hardcopy) or SK2T-2478 (CD-ROM) The intent of this technical library is to provide detailed information about each Open Blueprint component. The authors of these papers are the developers and designers directly responsible for the components, so you might observe differences in style, scope, and format between this paper and others.

Readers who are less familiar with a particular component can refer to the referenced materials to gain basic background knowledge not included in the papers. For a general technical overview of the Open Blueprint, see the *Open Blueprint Technical Overview*, GC23-3808.

Who Should Read This Paper

This paper is intended for audiences requiring technical detail about the Internationalization resource manager in the Open Blueprint. These include:

- Customers who are planning technology or architecture investments
- Software vendors who are developing products to interoperate with other products that support the Open Blueprint
- Consultants and service providers who offer integration services to customers

Contents

Figures

Internationalization Resource Manager

Introduction

There is an increased focus on worldwide software usage. Software may have to satisfy the language, culture, and character data encoding needs of a wide variety of users. While meeting this need, the software must fit or conform to the individual needs of each user.

The Internationalization resource manager component of IBM's Open Blueprint provides application enabling services for the management of internationalization resources in an open, distributed environment. Specifically, the Internationalization resource manager is one of the Application/Workgroup Services of the Open Blueprint as shown in the graphic on the cover of this paper.

The Internationalization resource manager is an application enabler. It utilizes the local system's services for the storage of the Internationalization resource manager managed data structures and utilizes distributed services and network services to support access to those data structures.

The Internationalization resource manager supports the portion of the application processing environment that deals with the language, culture, and character data encoding conventions. It also provides a set of standard programming interfaces to functions that use the information contained within the Internationalization resource manager managed data structures.

The Internationalization resource manager has dependencies upon other components of the Open Blueprint for the provision of services required to locate and access the Internationalization resource manager data structures in distributed environments.

In conjunction with a software development model, the Internationalization resource manager provides the means for developers to create internationalized software, where users can select the language, culture, and character data encoding of their choice.

Concepts

Software *internationalization* is a design methodology used to create software that is neutral with respect to the language, culture, and character data encodings that it is expected to support. The software produced can have its processing results modified at runtime, according to the language, culture, and character data encoding preferences of its users.

After the internationalized software has been developed, *localization* is used to provide the values necessary for a specific language, culture, and character data encoding. Localization information is made available to the software through a named data structure known as the *locale*.

The locale is created from source information that describes the local information necessary for the proper processing of data with respect to language, culture, and character data encoding. This locale information is a collection of values identifying the unique formatting specifications. These specifications are categorized according to the particular type of information they deal with, such as date and time or character classification.

The source definitions are stated in a symbolic manner and require the binding of the character encoding information to produce a usable locale data structure. The character encoding information maps the symbolic name of the character as used in the locale source file with the corresponding coded representation as used in the machine for which the data structure is being created.

The locale source information is bound with the character encoding information in a compilation process that produces the locale data structure. The data structure is also given a name as part of the compilation process. The name is provided in a recognized industry convention of a two-character language identifier, followed by a two-character country identifier and a character string identifying the character data encoding used. The International Organization for Standardization (ISO) standards for country names (ISO 3166) and language names (ISO 639) are used.

All of the locale creation activity must be performed prior to using the locale in the runtime environment. The locale and its name, which follows recognized industry conventions, provide all of the relevant information necessary to define the localized runtime environment. Providing a defined locale allows access by any number of applications and ensures consistent results.

The Internationalization resource manager provides a mechanism to describe and manage the various cultural elements of the internationalized processing environment expected by today's application users. It provides runtime access to localization information to the application through well defined interfaces, hiding any underlying implementation from the calling routines.

The programming model and related interfaces of the Internationalization resource manager have been defined through the Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interfaces (POSIX), X/Open Joint Internationalization Group (XoJIG) and UniForum (before the formation of the XoJIG) development efforts and those of the ANSI/ISO C programming language community. In addition to this work, IBM extensions are provided to the standard base support.

Background and Model

Evolution of Internationalized Applications

The requirement to create international software has caused a development methodology to evolve. This evolution is shown in Figure 1.

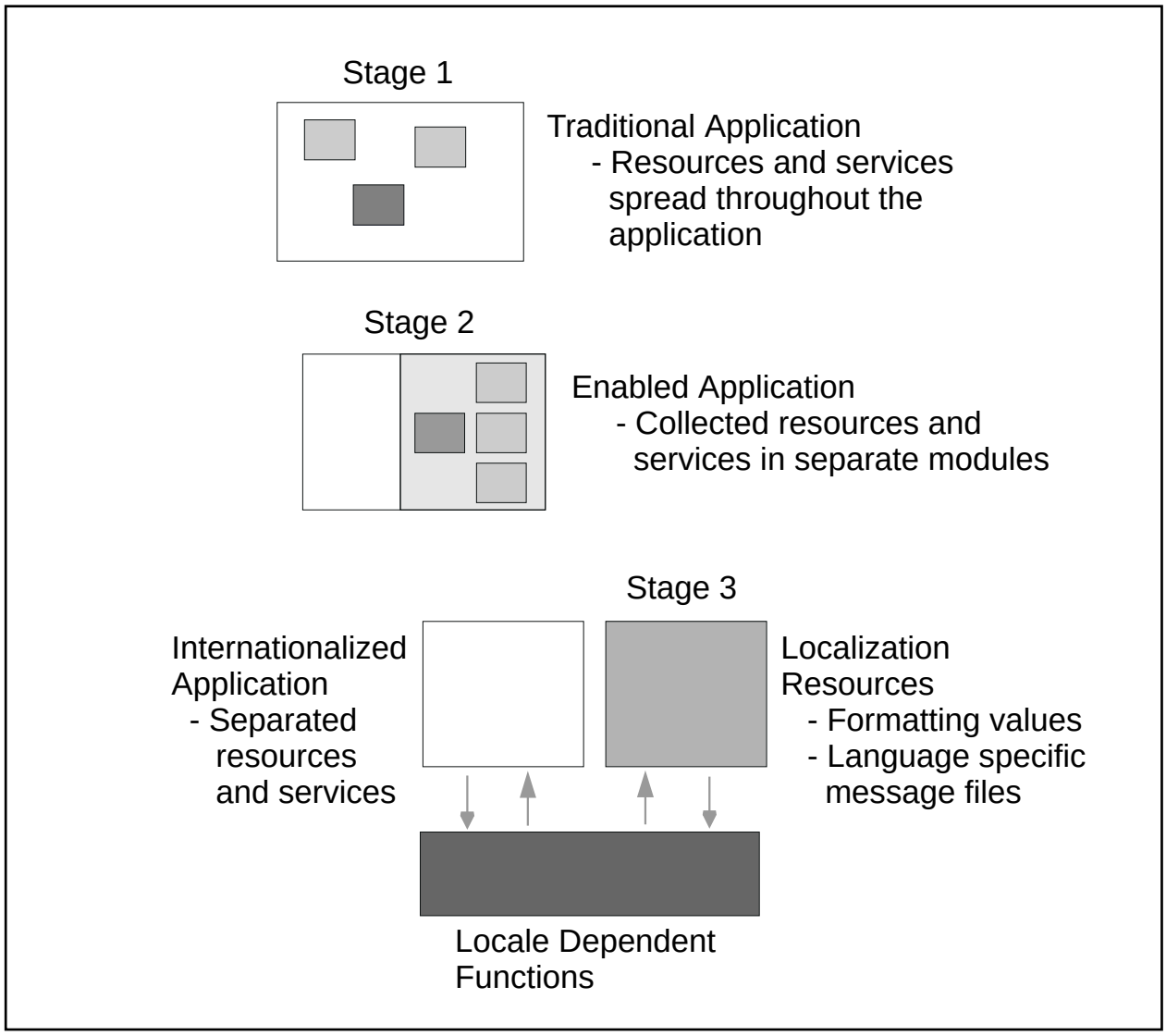


Figure 1. Evolution of International Software

Stage One: In this stage software functions providing support for, or having a sensitivity to, the language, culture, or character data encoding preferences were spread throughout the application. The assumption was that the processing environment of the software is the same as that of the developer. This method does not allow for any flexibility in supporting users with different sets of language, culture, and character encoding preferences. A new set of preferences has to be developed and compiled into each version of the software to satisfy unique requirements of the target users.

Software created with cultural support included at compile time does not allow for usage in cultures which have not been predetermined. The location of software usage and its intended users has to be known at design time.

Stage Two: The use of enabling techniques as described in the *National Language Design Guide, Volume 1*, promotes the collection of the language, culture, and character data encoding aspects of the software into modules. Packaging methodologies take advantage of the results of enabling and implementation changes to provide versions of the software that support a particular language, culture, and character encoding. The modularity reduces the maintenance effort for the software.

Stage Three: The software isolates the main application code from the resources and functions required for proper language, culture, and character data encoding support. This allows developers to concentrate more on the creation of desired software functions and less on the provision and management of the internationalization aspects. The set of local values necessary for language, culture, and character data encoding support are repackaged into localization resources known as locales. As much localization data as possible is stored externally from the application. Commonly used functions are collected into system services external to the application as well. Software is created with a dependency on these system services to provide the needed functions and resources for proper support of a user's expectations. Providing the necessary functions and resources by late binding of the localized information with the application is known as localization and is supported by the Internationalization resource manager on different platforms.

Internationalization and Localization

To exploit the availability of common services and resources for language, culture, and character data encoding, software must adhere to the concepts of internationalization and localization. Figure 2 on page 5 shows how these two concepts are related. By internationalizing the software for general worldwide use at build time and then localizing it for a particular language, culture, and character data encoding at run time, an application can be adapted to meet specific needs and preferences of a variety of users in different countries. This support has the goal of being able to allow any user, at any location, to use any application and have that application provide culturally-correct processing results.

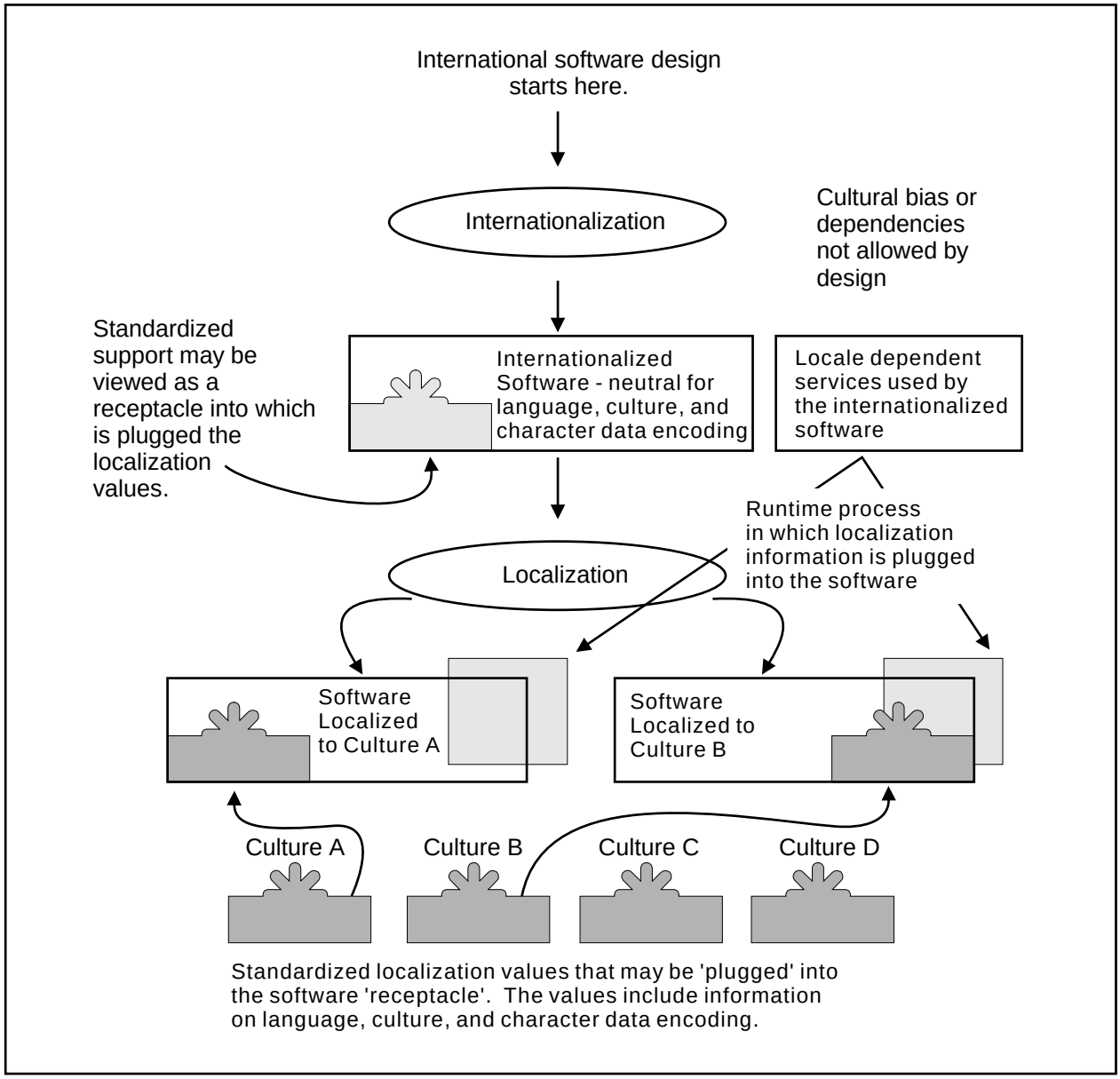


Figure 2. Internationalization and Localization. Internationalized software is incomplete without the associated localization.

Internationalizing software starts with careful design. Message libraries are created separately from the software which uses them. Formatting information describing local requirements is defined and stored separately from the application. The result is software that is neutral of any cultural bias or dependency. The plugs in the diagram represent the logical collection of the necessary localized information and message files. The result of this process is software using standard callable services for internationalization support. The software's environment is enabled, through the interfaces, to have its output dynamically altered at runtime by the user preferences. Software targeted for international usage should use this internationalization process as a design foundation.

Localization is best handled at execution time (late binding) to provide the most flexibility to the user of the application and to eliminate the costs of supporting multiple compiled versions. At runtime, a set of user preferences is logically “snapped into” the receptacles of the internationalized software. Since the software's processing results are dependent upon the localized values, and these can be controlled by the user, the user actually has control over the international support of the software.

Locales

A named locale defines and contains information for the application. The runtime support makes this information available to the application so the application data can be processed correctly according to the preferences of the user.

The locale model is a programming construct for the software developer. The application user should not normally have to deal with the locale. The construct has been used on various UNIX platforms, having its origins in the works of POSIX, X/Open and UniForum. This work has resulted in international standards recognition by ISO.

Locale Names: Some of the localization information is implicit in the name of the locale. In the Open Blueprint, locale names follow recognized industry conventions that are in various stages of standardization.

Locale names are composed of three separate terms as follows:

- Language identifier
- Country identifier
- Character data encoding identifier. This is also known as a code set identifier which is an identifier for the charmap resource.

An example of a locale name is *en_US.IBM-850* [1] where *en* is a two-character ISO 639 identifier for the language English, *US* is the two-character ISO 3166 identifier for the country USA and *IBM-850* is the character data encoding identifier which represents IBM's 850 code page or code set. Another example of a locale name is *FR_CA.ISO8859-1* where *FR* is a two-character identifier for the language French, *CA* is the two-character identifier for the country Canada and *ISO8859-1* is the character data encoding identifier which represents the ISO 8859-1 code page or code set.

The language and country identifiers follow their respective ISO standard formats. The entire three term name is under consideration for standardization in the recent draft X/Open document *Distributed Internationalization Services (DISS)*.

Specific locale names conforming to the conventions are entered into the system and maintained as part of the administration process.

Models described later in this document assume that the locale names and their respective contents have been defined and registered by some authority. This registration allows for the use of a *name* to provide the same results across platforms and systems. Without such constraints there can be no guarantee that the same name used on different systems will yield the same meaning.

Locale Content: The locale information as specified by POSIX and X/Open contains the following six standard categories:

Category	Description
LC_COLLATE	Character collation information
LC_CTYPE	Character classification attributes and case mapping information
LC_MONETARY	Monetary formatting specifications
LC_NUMERIC	Numeric data formatting specifications
LC_TIME	Date and time formatting specifications
LC_MESSAGES	Affirmative and negative response values

There are also two IBM extensions:

Category	Description
LC_SYNTAX	Variant character support within the locale
LC_TOD	Time and daylight savings time information

The IBM extensions provide support for the handling of variant characters in the EBCDIC environments where such characters are required for the syntax of the programming language being used with the underlying process code support, and time zone and daylight savings time information as well.

The locale categories are collections of similar information within the larger container known as the locale. The manner in which the system stores this localization information is an implementation issue for the system. Applications need not know the internal representation of the locale information because they do not deal with the contents directly.

Locale Structure Creation

A locale data structure is a prerequisite to the performance of any of the locale dependent services provided by the Internationalization resource manager. Locales, either as source and/or as compiled ready-to-use objects, may be shipped with base operating systems. Application development tools such as compilers may also ship locales. Additional locale structures may be created by taking the modified locale source definitions and character definitions (charmap resource) and processing them through a *locale definition compiler* (the localedef utility). The compiled output is a locale data structure used at runtime by the callable locale-dependent services coded within the application software. Figure 3 on page 8 illustrates how a locale is created.

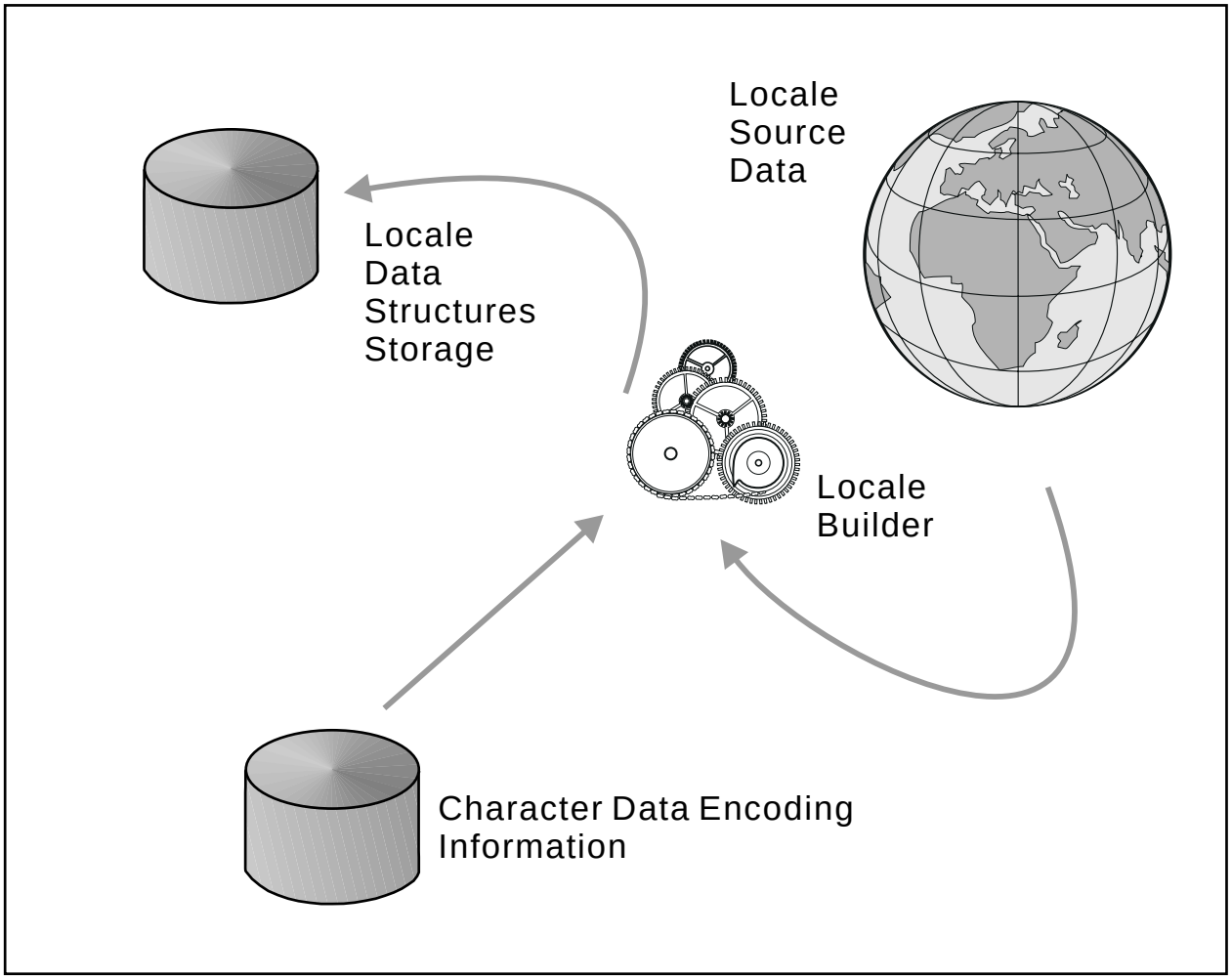


Figure 3. Locale Structure Creation

The locale source contains information describing the local values used to uniquely define the processing environment according to the software user's preferences. The source file has a unique identifier, the locale name. The character encoding information supplies the mapping between the symbolic information used in the locale source file and the machine representation of the characters. The locale builder combines this information into a structure that can be used on a specific system.

When locales are created, they are placed in a designated file directory.

POSIX or XPG4 Locale Model

The POSIX work was adopted by X/Open; the XPG4 locale model includes the POSIX locale model.

The foundation of this locale model is that one application equals one locale (for example, one language, country, and encoding combination per instantiation). This model allows for the setting of language, cultural, and encoding information at the process level through an opaque application global structure. It is also called a global model because locale changes are global, in that all threads within the process are affected. The model provides capabilities to set and query the current locale and locale categories. Applications therefore have the capability to switch locales in order to provide different cultural behavior.

Program Flow: The program flow for the global locale model can be broken down into the following steps:

1. Determine the currently active locale. This query helps to decide if the current locale is the right locale for the operations that are intended.
2. Change the locale to the desired locale if not already active.
3. Perform the intended operations
4. Restore the previous locale if it was changed

Runtime View: Figure 4 shows an overview of locale usage. The runtime locale elements in this figure correspond to the locale categories described under “Locale Content” on page 6.

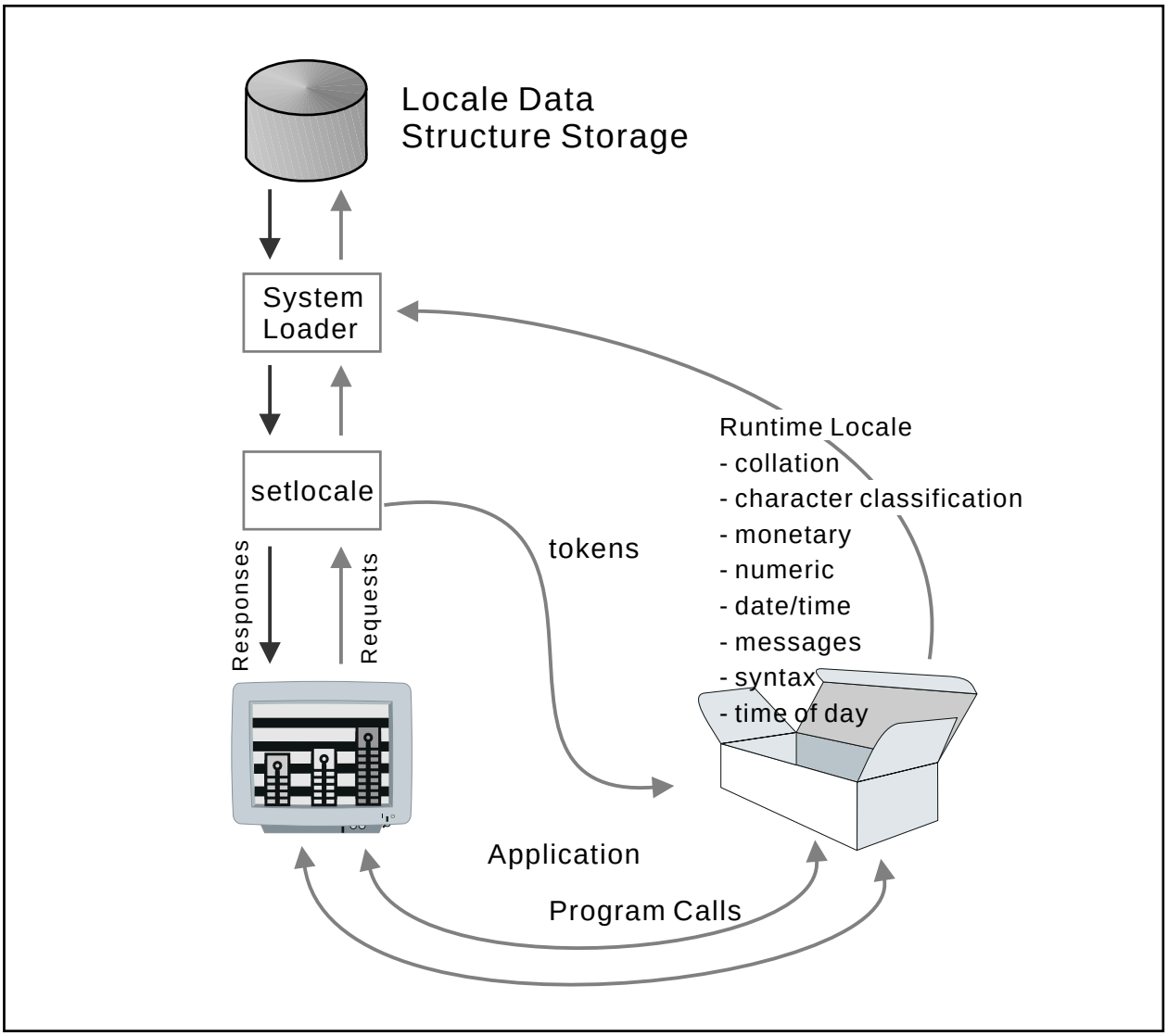


Figure 4. Runtime View

Current standards call for the use of the *C* or *POSIX* locale information to be used as the default value for locale settings.

Locale Initialization: The locale data structure created previously has been stored in the file system. The structure is retrieved from storage by the system loader based on a call initiated by the software wanting to use the localization information. This is done through the *setlocale* interface. Once loaded into storage, the locale's content is made available to the software through the locale-dependent services.

The setlocale service is used to initialize the locale data structure in storage. The application does not manage the international settings directly. It relies on the set of Internationalization resource manager services to do that work. The application can only call the provided services to interact with the locale data.

Extended Locale Model

The current global model provides a simple mechanism to write internationalized software. The model, however, is inadequate for software that needs to deal with multiple languages and cultural conventions at the same time, such as an internationalized spreadsheet program that supports different currencies and date formats for each cell. Multithreaded software, whether local or distributed, where two or more threads may be executing at a time, also challenges this model. In this case, if either thread changes the active locale, locale-sensitive operations in the other thread are also affected. Other limitations include locale synchronization and data representation in a distributed environment, context-sensitive rendering and text directionality, and the support of stateful encodings that allow a context to be maintained across different function calls.

There is work underway in the industry to define an extended locale model that would address these and other limitations of the global locales model. X/Open has produced such a document. It is a working draft called the *Distributed Internationalization Services Snapshot (DISS)*.

The extended locale model is similar to the POSIX locale. The extensions provide a new locale initialization function and support for multiple active runtime locales, each uniquely describing a set of localized information and addressable by the software.

Another example of an extended locale model is the Universal Language Support (ULS) which is implemented on OS/2.

Distributed Usage

When client software requests internationalization services that will be provided by a server, it is necessary that the locale used on the server be consistent with the expectations of the requester. For the internationalized environment to be replicated on the server side, the locale on the client needs to be uniquely and unambiguously identified. This is very important in order to allow locale replication across heterogeneous networks.

The software developer must design and implement methods of locale and encoding negotiation between the client and server sides. Assuming that this is done, a single set of localization resources is made available to the software users. A server process can only provide service to clients with the same localization requirement as the active locale. If a user request is for service associated with a locale other than the current active locale in the server, then the current active locale needs to be changed to the one requested. It is the server's responsibility to maintain the association of locales and user requests.

For a multithreaded server, serving multiple users with differing localization requirements, a specific server process for a specific locale can be designated. In this locale-per-server-process approach, it is imperative that the server process never changes its locale; the current active locale is the only active locale allowed for that server process. The onus is on the client code to direct the server requests to the

right server process. The server must block all other threads and access the correct locale during an operation that depends on that locale.

Note that the extended locale model provides for an object that can represent a well-known locale and can be used across a network. It is a handle to a locale structure which can be interchanged as an remote procedure call (RPC) object. This eases the enabling of consistent localized behavior in distributed software.

Resource Manager Definition

Internationalization Resource Manager

The Internationalization resource manager provides a mechanism to describe and manage the various elements of the internationalized processing environment expected by today's application users. This support is provided explicitly by APIs.

The Internationalization resource manager is based on open system standards as defined by POSIX, adopted by X/Open and documented in XPG4 (X/Open Portability Guides). The Internationalization resource manager also supports ANSI/ISO C and IBM extensions.

It supports a locale mechanism and related resources (formatting values and message files). This support acts as an application enabler, permitting the implementation of a user's specific language, cultural, and encoding needs. Associated with the resource manager is a locale definition compiler (the localedef utility).

The relationship to other selected components of the IBM Open Blueprint is also described.

The Internationalization resource manager is one of the Application Enabling Services components of the Open Blueprint. The Internationalization resource manager uses elements of the distributed and networking services components of the Open Blueprint to support resource access.

XPG4 Locale Services

The major functions of the Internationalization resource manager are as follows:

- Locale management
- Locale information retrieval
- Character classification
- Collation
- Character string handling
- Formatting
- Stream I/O
- Message files

Object-oriented (OO) style interfaces can be supported through the use of the class library provided interfaces. The Application Support Class[2] contains the basic data type classes using the POSIX and XPG4 programming model. They are dependent upon the locale sensitive functions of the C runtime.

Locale Management

Locale management deals with the retrieval of the locale data structures from the locale database (file system), the initialization of these structures into memory for application usage and the removal of these structures from memory when they are no longer needed. All of these actions are done from within the application software via callable services.

Locale initialization is handled by the following API:

Interface	Description
setlocale()	Sets, changes, queries locale categories or groups of categories, according to the values of the locale and category parameters specified

Locale Information Retrieval

These services provide access to specific values set within an instance of a locale. They only operate against the locale data structures in memory and not the locale data structures stored in the locale database (file system).

Retrieval of locale information is accomplished using the following APIs:

Interface	Description
nl_langinfo()	Retrieves the information associated with the specified locale item
localeconv()	Places values from the current locale into another structure for further use

Character Classification

Character classification based interfaces are used to determine the unique properties assigned to characters in the active locale. The characters have been assigned attributes or properties during the definition of the locale in the locale source file. Assignments include such attributes as “alphabetic.” Characters can be either common graphic characters as found in the file system (regular character-based) or they can also be defined as wide characters as used in the internal processing code. A wide character is defined as a character within a set of characters that contains all of the possible characters used in all of the supported locales. The wide character set can be viewed, from the encoding perspective, as a normalized character set. Typically the encoding used for the wide character set is different from the character encoding used in the file systems. Wide character data may be stored, but it is primarily used as a process code to eliminate the differences encountered when dealing with a variety of data encodings found in the file system.

Character classification services for regular character support are:

Interface	Description
isalnum()	Tests if this character is an alphanumeric character
isalpha()	Tests if this character is an alphabetic character
iscntrl()	Tests if this character is a control character
isdigit()	Tests if this character is a decimal digit (0....9)
isgraph()	Tests if this character is a graphic character
islower()	Tests if this character is a lowercase character
isprint()	Tests if this character is a printable character
ispunct()	Tests if this character is a punctuation character
isspace()	Tests if this character is a white space character (tab, space...)
isupper()	Tests if this character is an uppercase character
isxdigit()	Tests if this character is a hexadecimal character (0...9, a..f, A..F)

Note: The isblank() interface, which tests if a character is a blank, is an IBM character classification service for regular character support.

Character classification services for wide character support are:

Interface	Description
iswalnum()	Tests if this wide character is an alphanumeric character
iswalpha()	Tests if this wide character is an alphabetic character
iswcntrl()	Tests if this wide character is a control character
iswdigit()	Tests if this wide character is a decimal digit (0....9)
iswgraph()	Tests if this wide character is a graphic character
iswlower()	Tests if this wide character is a lowercase letter
iswprint()	Tests if this wide character is a printable character
iswpunct()	Tests if this wide character is a punctuation character
iswspace()	Tests if this wide character is a white space character (tab, space...)
iswupper()	Tests if this wide character is an uppercase character
iswxdigit()	Tests if this wide character is a hexadecimal character (0...9, a..f, A..F)
iswctype()	Tests if this wide character is a member of the character class specified
wctype()	General function to return a value used in the iswctype call

Note: The iswblank() interface, which tests if a wide character is a blank, is an IBM character classification service for wide character support.

Collation

Cultural based ordering means collating a set of characters according to a user's cultural conventions (or country standards). It is not the order that would result if the character data were ordered based on the binary values of the character codes themselves. For example, the same results will be achieved if a string is collated on an EBCDIC system or an ASCII system because the underlying binary coding values are not used.

Collation based services are:

Interface	Description
strcoll()	Compares two-character strings according to their collation weights
strxfrm()	Transforms each character of the provided string into its respective collation weight
wcscoll()	Compares two-character strings according to their collation weights
wcsxfrm()	Transforms a wide character string into an array of representative collation weights

The following IBM collation services have also been added:

Interface	Description
collequiv()	Returns a list of equivalent collating elements for the primary weight given
collorder()	Returns a list of collating elements
collrange()	Calculates the range list of collating elements between the two primary collating weights given
colltostr()	Converts a collating element to a string

Character String Handling

These services are used to manipulate character data strings. Both the process code format and the file code formats are supported. An example of string handling is a concatenation operation where two strings are joined.

Character based string handling services are:

Interface	Description
strcat()	Appends a copy of the input string to the end of the target string
strchr()	Searches a target string for the occurrence of the specified string
strcmp()	Compares two-character strings
strcpy()	Copies the content of one string into another string
strcspn()	Determines the number of characters in the initial segment of a target string that do not appear in the source string
strlen()	Calculates the length of a character string
strncat()	Appends a specified number of characters from the source string to the target string
strncmp()	Compares a specified number of characters in the source string to the target string
strncpy()	Copies a specified number of characters from the source to the target string
strpbrk()	Locates the first occurrence in the target string of any character from the source string
strrchr()	Locates the last occurrence of a specified character in the target string
strspn()	Calculates the number of characters in the target string which consists entirely of characters from the source string
strstr()	Finds the first occurrence of the target string in the source string
strtok()	Sequence of calls to wcstok will break the target string into a sequence of tokens, each of which is delimited by characters from the source string

Wide character based string handling services are:

Interface	Description
wcscat()	Appends a copy of the input string to the end of the target string
wcschr()	Searches a target string for the occurrence of the specified string
wcscmp()	Compares two wide character strings
wcscpy()	Copies the content of one string into another string
wcscspn()	Determines the number of wide characters in the initial segment of a target string that do not appear in the source string
wcslen()	Calculates the number of wide characters in a string
wcsncat()	Appends a specified number of wide characters from the source string to the target string
wcsncmp()	Compares a specified number of wide characters in the source string to the target string
wcsncpy()	Copies a specified number of wide characters from the source to the target string
wcspbrk()	Locates the first occurrence in the target string of any character from the source string
wcsrchr()	Locates the last occurrence of a specified character in the target string
wcsspn()	Calculates the number of wide characters in the target string which consists entirely of characters from the source string
wcstok()	Sequence of calls to wcstok will break the target string into a sequence of tokens, each of which is delimited by a wide character from the source string
wcswcs()	Locates the first occurrence in the target string of a sequence of wide characters specified in the source string
wcswidth()	Determines the number of display positions required to display the wide character string on a display device
wcwidth()	Determines the number of display positions required to display the wide character on a display device

In addition, XPG4 includes the ANSI C interfaces that support conversions between multibyte character streams and internal/processing (normalized character set) wide character streams:

Interface	Description
mblen()	Determines the length in bytes of the multibyte character provided
mbstowcs()	Converts a specified number of multibyte characters to their corresponding wide character representations
mbtowc()	Converts the multibyte character to a wide character
wcstombs()	Converts a string of wide characters to their corresponding multibyte character representation
wctomb()	Converts the wide character specified into a multibyte character

The XPG4 standard can handle a variety of character data such as single, double, and multibyte character sets. However, it does not support stateful encodings (where a sequence of operations is based on a previously established condition or state). Thus, extensions based on the ISO/IEC 9899:1994, are supported to provide shift-in/shift-out stateful encoding for EBCDIC double-byte character data:

Interface	Description
mbrlen()	Determines the length in bytes of the multibyte character provided, when using stateful encoding
mbrtowc()	Converts the multibyte character in a stateful encoding to a wide character
wcrtomb()	Converts the wide character specified into a multibyte stateful encoded character
mbsrtowcs()	Converts a specified number of multibyte characters in a stateful encoded format to their corresponding wide character representations
wcsrtombs()	Converts a string of wide characters to their corresponding multibyte character stateful encoding representation

Formatting

Formatting services support the requirement to format character strings for date, time, and monetary information according to the cultural preferences of the user or to meet country standards.

Formatting services are:

Interface	Description
strftime()	Formats date and time values into a multibyte character string
wcsftime()	Formats date and time values into a wide character representation
strptime()	Converts a character string into date and time values using the specified formats
strfmon()	Converts a monetary value to a formatted string of characters

Stream I/O

Stream I/O services support the requirement to access multibyte or wide character data.

Stream I/O services are:

Interface	Description
fgetwc()	Gets a multibyte character from the specified stream
fgetws()	Gets a multibyte character string from the specified stream
getwc()	Gets a multibyte character from the standard input stream
putwc()	Outputs a wide character to the specified stream
putws()	Outputs a wide character string to the specified stream
ungetwc()	Pushes the specified wide character back onto the specified stream

Message Files

These are the services necessary to manage the various libraries of translated textual information required by the applications. They provide support necessary to keep textual information related to the application separate from the application's executable code and to perform the proper information substitution at runtime.

These message handling services are outside of the normal locale semantics in that they are not contained within the locale data structure. However, they operate within the locale mechanism. They are extremely important for the support of translated textual information within the internationalized software.

Message handling services are:

Interface	Description
catopen()	Opens the specified catalog and return a catalog descriptor for subsequent use by the catgets function
catclose()	Closes a catlog resource made available by the catopen function
catgets()	Retrieves the indicated message from the specified message catalog

There is one message handling utility:

Interface	Description
gencat	Creates the catalog file from the message file source

There are two additional IBM message handling utilities:

Interface	Description
runcat	Generates the header file consisting of numbers for the mnemonics and the catalog file consisting of messages with a set and message numbers from a message file source
mkcatdefs	Generates the message header file consisting of numbers for the mnemonics

Object-Oriented Support

Internationalization support is provided in the Application Support Class by the IString, ITime, and IDate set of functions. IString is used for character string handling in single, double and multibyte encodings. ITime provides time formatting and manipulation functions. IDate is used for date formatting and manipulation. All member functions of the above classes rely on the current locale for localization information.

In the future, C++ programs will be able to encapsulate cultural differences including internationalization support for character classification, collation, character string handling, formatting and parsing of numeric, monetary, and date/time values, and message retrieval. This is described in the *Draft Proposed International Standard for Information Systems - Programming Language C++*.

IString

IString is a complex class, providing a comprehensive set of functions for string manipulation and support for the single-byte character set (SBCS), the double-byte character set (DBCS), and the multibyte character set (MBCS). It too uses the XPG4 programming model and support from the C runtime.

For binary, character, hexadecimal, and decimal conversions use:

b2c Converts a string of binary digits to a normal string of characters

b2d Converts a string of binary digits to a string of decimal digits

b2x Converts a string of binary digits to a string of hexadecimal digits

c2b Converts a string of characters to a string of binary digits

c2d Converts a normal character string to a string of decimal digits

c2x Converts a normal character string to string of hexadecimal digits

x2b Converts a string of hexadecimal digits to a string of binary digits

x2c Converts a string of hexadecimal digits to a normal character string

x2d Converts a string of hexadecimal digits to a string of decimal digits

d2b Converts a string of decimal digits to a string of binary digits

d2c Converts a string of decimal digits to a normal character string

d2x Converts a string of decimal digits to a string of hexadecimal digits

For character string editing use the functions:

center
Centers the receiver within a string of a specified length

change
Changes occurrences of a specified pattern to a specified replacement string

copy
Replaces the receiver's contents with a specified number of replications of itself

insert
Inserts the specified string after the specified location

leftJustify
Left justifies the receiver in a string of a specified length

lowerCase
Translates all of the uppercase letters in the receiver to lowercase

overlayWith
Replaces a specified portion of the receiver's contents with the specified string

remove
Deletes the specified portion of the string

reverse
Reverses the receiver's contents

rightJustify
Right justifies the receiver in a string of the specified length

strip
Strips both the leading and trailing characters or character

stripBlanks
Strips both leading and trailing white space (blanks)

stripLeading
Strips the leading character or characters

stripTrailing
Strips the trailing character or characters

translate
Converts all of the characters that are in the first specified string into the corresponding characters in the second string

upperCase
Translates all lowercase characters in the receiver to uppercase

Forward searching permits strings to be searched in a number of ways. The search start position can be indicated, otherwise it will default to the beginning of the string. Function provided is:

indexOf
Returns the byte index of the first occurrence of the specified string within the receiver

For character string manipulation use:

operator+
Concatenates two strings

operator+=
Concatenates the specified string to the receiver and replaces the receiver

operator=
Replaces the content of the string

For character testing use the following functions. Basically they are used to determine if an IString is a member of a specific character set.

disableinternationalization
Disables locale based string operations

enableinternationalization
Enables locale based string handling

includesDBCS
If any character is DBCS, true is returned

includesMBCS
If any characters are MBCS, true is returned

includesSBCS
If any characters are SBCS, true is returned

isDBCS
If all characters are DBCS, true is returned

isMBCS
If all characters are MBCS, true is returned

isSBCS
If all characters are SBCS, true is returned

isValidDBCS
If no DBCS characters have a second byte of 0, true is returned

isValidMBCS
If no MBCS characters have a second byte of 0, true is returned

ITime

ITime class represents units of time as portions of days and provides support for converting these units of time into numeric and character string format. Comparisons and other operations may be performed on ITime objects by adding them to or subtracting them from other ITime objects.

Comparison functions are:

operator!=
Compares two time objects to determine whether they are not equal

operator<
Compares two objects to determine whether one is less than the other

operator<=
Compares two objects to determine whether one is less than or equal to the other

operator==
Compares two objects to determine whether they are equal

operator>
Compares two objects to determine whether one is greater than the other

operator>=
Compares two objects to determine whether one is greater than or equal to the other

Constructors can be used to return the current time, copy another time object, give the time difference between midnight and time value provided.

To obtain the current time, use the following:

now()
Returns the current time, can be used as a constructor

Manipulation of time object values is done using:

operator+
Adds two objects

operator+=
Adds two objects and stores the results in the receiver

operator-
Subtracts one object from another

operator-=
Subtracts two objects and stores the results in the receiver

For time queries, that is to access the hours, minutes, seconds of a time object, use:

asICnrTime()
Returns the time as a container ICnrTime structure

asSeconds()
Returns the number of seconds since midnight

hours()
Returns the number of hours past midnight

minutes()
Returns the number of minutes past the hour

seconds()
Returns the number of seconds past the minute

To represent the time value as an IString representation and output to a stream, use:

asString
Returns the ITime object as a string formatted according to the format specifiers

IDate

IDate class represents specified dates. It provides general day and date handling functions to work with year, month, and day values. This class returns language sensitive information such as the names of days of the week and months in the language of the user's system.

Date comparison functions are:

operator!=
If the IDate objects represent different dates, true is returned

operator<
If the left hand operand represents a date prior to that of the right hand operand, true is returned

operator<=
If the left hand operand represents a date prior to or identical to that of the right hand operand, true is returned

operator==
If the IDate objects represent the same date, true is returned

operator>
If the left hand operand represents a date subsequent to the date represented by the right hand operand, true is returned

operator>=
If the left hand operand represents a date subsequent to or identical to the right hand operand, true is returned

Constructors can be used to return the current day, copy another IDate object, give the year, month and day or year and day for a given day, return the current day, or the Julian day number.

To obtain the current date, use the following:

today()
Returns the current date, can be used as a constructor

Manipulation of date object values is done using:

operator+
Adds an integral number of days to the left hand operand to yield a new IDate

operator+=
Adds an integral number of days to the left hand operand and stores the result in that operand

operator-
Subtracts an integral number of days from the left hand operand to yield a new IDate, if the right hand operand was also an IDate, the difference between the two dates is given

operator-=
Subtracts an integral number of days from the right hand operand and places the result in that operand

For general date queries, use:

dayName
Returns the name of the day

daysinMonth
Returns the number of days in the specified month of the specified year

daysinYear
Returns the number of days in the specified year

monthName
Returns the name of the month

monthofYear
Returns an index of the specified month

year
Returns the year

To perform validation on date based information use:

isLeapYear If the specified year is a leap year, true is returned

isValid Indicates whether the specified date is valid

To represent the IDate as an IString representation and output to a stream, use:

asString
Returns the IDate object as a string formatted according to the format specifiers

Extended Locale Services

The extended locale services are a potential future enhancement to the Internationalization resource manager. They are the subject of a working draft from X/Open called the *Distributed Internationalisation Services Snapshot (DISS)*.

The services differ from those previously stated for the POSIX locale in that they allow for the locale name to be passed as a parameter of the call. This allows for the specification of the localization information required to be directly associated with the operation requested. This package can then be handled anywhere as a complete unit of work.

In the ULS implementation on OS/2, which is another example of extended locale model support, UniCreateLocaleObject allocates system resources associated with a specified locale and provides the caller with a locale object handle. This locale object handle is used by the other functions to control their locale behavior for the duration of the function call. An application is allowed to create multiple locale object handles. Each locale object handle may refer to a different locale, thus allowing the application to take advantage of multiple and concurrent locale support.

Relationships with Other Open Blueprint Resource Managers

The graphic on the cover of this paper shows the overall schematic of the Open Blueprint model. The Internationalization resource manager is a part of the application enabling services component and is dependent upon other components of the Open Blueprint for services in support of its role as a resource manager.

Local Operating System Services

Locally, the Internationalization resource manager will make extensive use of the local operating system services to perform dynamic loading of the locale data structures, to perform task management and scheduling as well as memory management and termination (cleanup) services. The replication of internationalization services across platforms will enhance the portability of internationalized software.

Security Resource Managers

Security services are used to authenticate calls to access locale data structures. Locales can be classed as either public or private with necessary levels of access control in place to regulate their use. Software should not share localization resources unless specifically authorized. Access controls for custom locales is a must, while default locales should have free access to encourage their use.

Directory Resource Manager

Directory services are used to locate localization resources not available on the local system. A locale initialization request may cause the fetching of a locale from a repository of locales on a remote server. The only impact, in this case, should be the network delay and not a failure in the load request. The actual location of the locale data structure should be transparent to the requester.

File Resource Manager

The distribution of localization information might not be pervasive. There may be a need to have infrequently used resources maintained on selected servers and allow them to send out the requested data as needed. In such cases requests to the Internationalization resource manager would cause a related request to the distributed file system to send the necessary resources.

---

1 Mixed case locale names are often used on target systems that support case sensitive file names.

2 The Application Support Class is part of the IBM Open Class Library.

Bibliography

IBM *National Language Design Guide (NLDG)*

Volume 1, Designing Enabled Products: Rules and Guidelines, SE09-8001-01

Volume 2, National Language Support Reference Manual, SE09-8002-01

To order (in the USA), call 1-800-879-2755

The Open Software Foundation *Internationalization Made Easy*, A White Paper, OSF-D-WP5-0790-1, September 1990

POSIX 1003.2 *Information Technology, Portable Operating System Interfaces (POSIX) - Part 2: Shell and Utilities*, The Institute of Electrical and Electronic Engineers, Inc.

SHARE, Inc. Report SSD No. 366 *ASCII and EBCDIC Character Set and Code Issues in Systems Application Architecture*, The ASCII/EBCDIC Character Set Task Force. Edited by Edwin Hart, The Johns Hopkins University, Applied Physics Laboratory, Laurel, Maryland, USA Published by SHARE Inc., 111 East Wacker Drive, Chicago, Illinois, USA 60601; June 1989.

SHARE Europe (SEAS) White paper *National Language Architecture*. Edited by Klaus Daube, Oerlikon Bührle RZ AG, Zürich Switzerland. Published by SHARE Europe Headquarters, 17, Rue Pierres-du-Nitron, CH-1207 Geneva, Switzerland; June 1990.

SHARE Europe Association *White paper on national character, language and keyboard problems*. National Character Task Force. Published by SHARE Europe Headquarters, 17, Rue Pierres-du-Nitron, CH-1207 Geneva, Switzerland; September, 1985

X/Open Company Limited X/Open Portability Guides. *Internationalisation Guide*, Version 2, (ISBN: 1-85912-002-4)

X/Open Company Limited *Distributed Internationalisation Services X/Open Snapshot*, Version 2, (ISBN: 1-85912-033-4)

Working Paper for Draft Proposed International Standard for Information Systems - Programming Language C++ Document number: X3J16/96-0018 WG21/N0836

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM
IBMLink
Open Blueprint
OS/2

The following terms are trademarks of other companies:

POSIX	Institute of Electrical and Electronic Engineers
UniForum	UniForum Association
X/Open	X/Open Company Limited

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Communicating Your Comments to IBM

If you especially like or dislike anything about this paper, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply. Feel free to comment on specific error or omissions, accuracy, organization, subject matter, or completeness of this paper.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send comments by FAX, use this number:

  United States and Canada: 1-800-227-5088.

- If you prefer to send comments electronically, use one of these ID's:
  - Internet: **USIB2HPD@VNET.IBM.COM**
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**

Make sure to include the following in your note:

- Title of this paper
- Page number or topic to which your comment applies

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

G325-6583-00