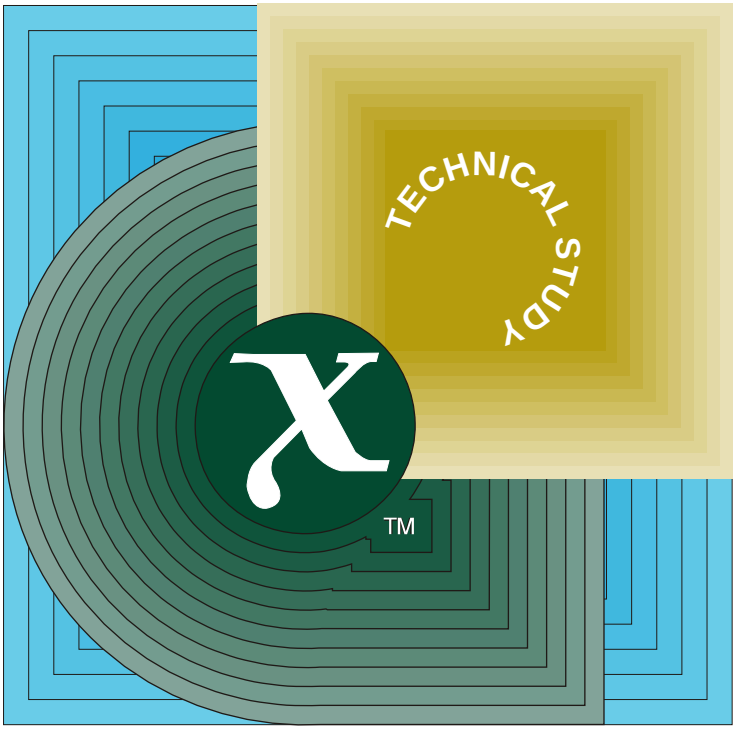
Technical Study

Desktop Internationalization



THE Open GROUP

[This page intentionally left blank]

X/Open Technical Study

Desktop Internationalisation

X/Open Company Ltd.

X/Open Technical Study

Desktop Internationalisation

X/Open Document Number: E501

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.org

Contents

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported standard. In addition, they can demonstrate compliance with the majority of X/Open CAE specifications once these specifications are referenced in an X/Open component or profile definition and included in the X/Open branding programme.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision, in this case as the subject technology develops and to align with emerging associated international standards. X/Open makes a distinction between revised specifications which are fully backward compatible and those which are not:

- a new *Version* indicates that this publication includes all the same (unchanged) definitive information from the previous publication of that title, but also includes extensions or additional information. As such, it *replaces* the previous publication.

- a new *Issue* does include changes to the definitive information contained in the previous publication of that title (and may also include extensions or additional information). As such, X/Open maintains *both* the previous and new issue as current publications.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done in any one of the following ways:

- anonymous ftp to ftp.xopen.org
- ftpmail (see below)
- reference to the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information using ftpmail, send a message to ftpmail@xopen.org with the following four lines in the body of the message:

```
open
cd pub/Corrigenda
get index
quit
```

This will return the index of publications for which Corrigenda exist. Use the same email address to request a copy of the full corrigendum information following the email instructions.

This Document

This document is a Technical Study of internationalisation aspects of X/Open Common Desktop Environment (XCDE) specifications.

Computer systems and applications are increasingly expected to work in an international environment in which different languages, character sets and cultural conventions are in use. This poses a number of requirements. The growth of distributed computing, with systems and applications interworking across networks, is making these requirements more urgent. At the same time, they affect the networking technology on which distributed systems are based; if it does not take them into account, the ability of a distributed system to work in different language and cultural environments is limited.

This Technical Study identifies the implications of internationalisation requirements on the specifications of the X/Open Common Desktop Environment (XCDE).

The X/Open XCDE specifications considered are the following (full details are given in the list of references):

- X Window System Protocol
- Xlib
- X Toolkit Intrinsics
- Motif Toolkit API
- File Formats and Application Conventions

- CDE Definitions and Infrastructure
- CDE Services and Applications
- Calendaring and Scheduling API.

This document is structured as follows:

- Chapter 1 on page 1 discusses internationalisation in more detail, and in particular describes the provisions that are made for it in international standards and in the X/Open Common Applications Environment.
- Chapter 2 on page 11 describes the XCDE as a whole.
- Chapter 3 on page 15 and Chapter 4 on page 27 discuss the individual specifications listed above and the internationalisation issues associated with them.
- Chapter 5 on page 49 contains conclusions and recommendations.

Trade Marks

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

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of X/Open Company Limited.

X/Open acknowledges that there may be other products that might be covered by trade mark protection and advises the reader to verify them independently.

Referenced Documents

The following X/Open documents are referenced in this Technical Study:

Distributed Internationalisation Framework
: X/Open Snapshot, January 1995, Distributed Internationalisation Framework (ISBN: 1-85912-079-2, S503).

Distributed Internationalisation Services
: X/Open Snapshot, December 1994, Distributed Internationalisation Services, Version 2 (ISBN: 1-85912-033-4, S308).

Internationalisation Guide
: X/Open Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304).

Layout Services
: X/Open Snapshot, December 1994, Portable Layout Services: Context-dependent and Directional Text (ISBN: 1-85912-075-X, S425).

Locale Registry
: X/Open Electronic Publication, October 1993, Locale Registry Procedures (ISBN: 1-872630-94-4, G303).

Migration Guide
: X/Open Guide, July 1992, XPG3-XPG4 Base Migration Guide (ISBN: 1-872630-49-9, G204).

Motif Toolkit API
: X/Open CAE Specification, April 1995, Motif Toolkit API (ISBN: 1-85912-024-5, C320).

UCS
: X/Open Technical Study, February 1994, Universal Multiple-Octet Coded Character Set Coexistence and Migration (ISBN: 1-85912-031-8, E401).

UTF-8
: X/Open CAE Specification, April 1995, File System Safe UCS Transformation Format (UTF-8) (ISBN: 1-85912-082-2, C501).

X11R5 File Formats
: X/Open CAE Specification, May 1995, Window Management (X11R5): File Formats and Applications Conventions (ISBN: 1-85912-090-3, C510).

  This comprises:

  — Inter-Client Communications Conventions Manual (ICCCM)

  — X Logical Font Description (XLFD)

  — Compound Text

  — Bitmap Distribution Format (BDF).

X11R5 X Protocol
: X/Open CAE Specification, May 1995, Window Management (X11R5): X Window System Protocol (ISBN: 1-85912-087-3, C507).

X11R5 X Toolkit
: X/Open CAE Specification, May 1995, Window Management (X11R5): X Toolkit Intrinsics (ISBN: 1-85912-089-X, C509).

X11R5 Xlib
: X/Open CAE Specification, May 1995, Window Management (X11R5): X Lib - C Language Binding (ISBN: 1-85912-088-1, C508).

XCDE: Definitions and Infrastructure
: X/Open CAE Specification, April 1995, X/Open Common Desktop Environment (XCDE): Definitions and Infrastructure (ISBN: 1-85912-070-9, C324).

XCDE: Services and Applications
: X/Open CAE Specification, April 1995, X/Open Common Desktop Environment (XCDE): Services and Applications (ISBN: 1-85912-074-1, C323).

XCS
: X/Open CAE Specification, April 1995, Calendaring and Scheduling API (XCS) (ISBN: 1-85912-076-8, C321).

XCU, Issue 4
: X/Open CAE Specification, July 1992, Commands and Utilities, Issue 4 (ISBN: 1-872630-48-0, C203).

XPG4, Version 2
: The X/Open Branding Programme, How to Brand — What to Buy, February 1995 (ISBN: 1-85912-084-9, X951).

XSH, Issue 4
: X/Open CAE Specification, July 1992, System Interfaces and Headers, Issue 4 (ISBN: 1-872630-47-2, C202).

The following non-X/Open documents are referenced in this Technical Study:

ANSI C
: American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.64-1979 C
: ANSI standard X3.64-1979: Additional Controls for Use with American Standard Code for Information Interchange.

CCITT T.100
: CCITT Recommendation T.100: 1984, International Information Exchange for Interactive Videotex.

CCITT T.61
: CCITT Recommendation T.61: 1984, Character Repertoire and Coded Character Sets for the International Teletex Service, Geneva, 1980, amended at Malaga-Torremolinos, 1984.

ISO 2022
: ISO 2022: 1986, Information Processing — ISO 7-bit and 8-bit Coded Character Sets — Coded Extension Techniques.

ISO 2375
: ISO 2375: 1985 Data Processing — Procedure for Registration of Escape Sequences.

ISO 3166
: ISO 3166: 1988, Codes for the Representation of Names of Countries, Bilingual edition.

ISO 639
: ISO 639: 1988, Codes for the Representation of Names of Languages, Bilingual edition.

ISO 8859-1
: ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 1: Latin Alphabet No. 1.

ISO C
: ISO / IEC 9899: 1990, Programming Languages — C (technically identical to ANSI standard X3.159-1989).

ISO / IEC 10646
: ISO / IEC 10646: 1993, Information Technology — Universal Multiple-Octet Coded Character Set (UCS).

ISO / IEC 6429
: ISO / IEC 6429: 1992, Information Technology — Control Functions for Coded Character Sets.

ISO / IEC 646
: ISO / IEC 646: 1991, Information Processing — ISO 7-bit Coded Character Set for Information Interchange.

MSE
: ISO / IEC 9899: 1990 / Amendment 1: 1994, Multibyte Support Extensions (MSE) for ISO C.

POSIX.1
: IEEE Std 1003.1-1988, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.2
: IEEE Std 1003.2-1992, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities.

RFC 822
: Internet RFC 822 — Standard for the Format of ARPA Internet Text Messages, the Internet Architecture Board, 1982.

RFCs 1521 and 1522
: Internet RFCs 1521 and 1522 — MIME (Multipurpose Internet Mail Extensions) — Parts 1 and 2, the Internet Architecture Board, 1993.

UNICODE Standard
: The Unicode Consortium, The Unicode Standard, Worldwide Character Encoding Version 1.0, Volume One, Addison-Wesley, 1991.

Chapter 1

Internationalisation

1.1 Introduction

Computer systems must meet the needs of users who speak different languages, conform to different cultural conventions and follow different business practices. This means that the facilities of the X/Open Common Applications Environment (CAE) must not impose constraints on the users' languages, cultural conventions or business practices, and must include facilities that support the development of applications that can be used in multiple language, cultural and business environments.

Understanding of the implications of this has evolved as the X/Open CAE has developed. It is evolving still.

The most obvious area in which constraints can be imposed, and the area that has received the most attention, is that of character sets and their encodings. But programs have often imposed other constraints by making assumptions about:

- directionality (whether text is written from right to left or from left to right)
- collation rules used in comparing, ordering and sorting character strings
- rules for character classification into categories such as alphabetic, numeric, punctuation, and so on
- shift rules for character case conversion
- the way in which numbers are written (for example, the use of a comma (,) or period (.) as decimal separator)
- the value and positioning of the currency symbol
- the way in which dates are written (for example, dd/mm/yy or mm/dd/yy, or using Asian formats with dissimilar date component separators)
- the way in which times are written (for example, 10:24 PM, 22.24, 10h24)
- the use of upper and lower-case characters
- the language of the user interface (for example, error messages in a particular natural language have often been hard-coded into a program).

This chapter summarises the provisions that have been made in international standards and in the X/Open CAE for addressing internationalisation issues. The international standards concerned fall into two categories. The first is that of standards for character sets and encodings used in data communication. The second is that of standards for information processing; specifically, the C programming language and the POSIX operating system interface. X/Open publications have always taken account of developments in international standards, and have often anticipated and influenced them. This chapter therefore concludes with an indication of the current direction of internationalisation work within X/Open.

1.2 Character Sets and Encodings

A wide variety of character sets is used to represent the languages of the world. This report is written in the English language, represented using the characters of the basic Latin alphabet. Other Western European languages are represented using character sets that include those of the basic Latin alphabet plus a few additional characters (different additional characters are used by each language). Other languages (such as Greek and Russian) use character sets that are alphabetic but are not variants of the Latin alphabet. Yet other languages, such as Japanese and Chinese, use ideographic scripts that are not alphabetic. Mathematical and scientific text, in any language, uses characters borrowed from several different alphabets.

When held in computer storage, and while being transmitted between computers, characters are encoded as bit patterns. The bit patterns that constitute the encodings of a character set are called a codeset.

A number of encoding schemes used to represent characters being transmitted between computers have been standardised by national standards bodies, the CCITT and ISO. In each of these standards, a character is typically encoded as one or more octets, where an octet is a sequence of 8 bits, each of which can take the value 0 or 1.

Early communication protocols were designed for communication over low bandwidth lines and with relatively ''dumb'' devices such as teletypes. They used the minimum possible number of bits per character, and distinguished between graphic characters, which would be printed, and control characters, which would affect the operation of the remote device. Control characters included characters used to control communication (such as the <SOH> character that indicated the start of header information) and also characters used to control printing (such as the <CR> character that, on a teletype, caused the carriage to return to its starting position).

These facts affected the character encoding schemes that were used in conjunction with early protocols. The American Standard Code for Information Interchange (ASCII) was directly descended from such schemes. It is the basis of ISO/IEC 646, has considerably influenced later schemes, and is still in use. It represents the basic Latin alphabet, plus some additional control characters, using 7 bits per character. Control characters are encoded with values in the range 00 to 1F (hexadecimal).

Modern communication protocols, including the OSI protocols standardised by ISO and the protocols of the Internet protocol suite, transport 8-bit data transparently. This allows the use of encoding schemes that use all 8 bits of an octet and that do not reserve particular values for protocol control purposes.

A mechanism, intended for use with 7-bit or 8-bit encoding schemes, by which several different schemes can be used within a single transmission, is defined in ISO 2022. In this mechanism, certain control characters perform a *shift* function which determines how subsequent codes are to be interpreted. (This is by analogy with a typewriter, on which the <Shift> keys determine the symbols that will be printed when other keys are subsequently pressed.) The mechanism also allows the possibility that the encoding of a character can occupy more than one octet. Essentially, the *unshifted* codes represent the characters of the basic Latin alphabet, while *shifted* codes represent the characters of some other character set (as agreed by the communicating parties). With multiple-octet-per-character encoding schemes, any character set can be encoded.

A register of character sets and encodings is defined in ISO 2375. Encodings for most Western European character sets and for Japanese Kanji are registered.

Encodings compatible with ISO 2022 for the character sets of most languages used in Europe and North America (including Greenlandic, Russian and Turkish) and also of Afrikaans, Arabic, Esperanto and Hebrew, are defined in ISO 8859.

Encoding schemes that use the mechanism of ISO 2022 have been standardised for use in the Teletex service (see the T.61 CCITT Recommendation) and the Videotex service (see the T.100 CCITT Recommendation).

It should be noted that all the above standards use the same encodings as ISO / IEC 646 for the characters of the basic Latin alphabet. They also maintain the principle, even in multi-octet encodings, that octets in the range 00 to 1F (hexadecimal) are reserved for control characters.

However, the latest encoding standard, ISO / IEC 10646 (which incorporates the work of the UNICODE consortium), departs from these principles.

ISO / IEC 10646 is intended to cover the character sets of all languages that may be used in conjunction with computer systems. It defines a four-octet representation for each character. The characters whose representations have zero as their two most significant octets form what is known as the Basic Multilingual Plane (this includes most alphabetic character sets). Two forms of encoding are permitted:

UCS-2 This form applies where only characters in the Basic Multilingual Plane are used. In it, the encoding of a character consists of the two least significant octets of its four-octet representation.

UCS-4 This form permits the encoding of any character. In it, the encoding of a character consists of the whole of its four-octet representation.

In addition to the UCS-2 and UCS-4 forms, ISO / IEC 10646 allows a composite graphical symbol to be represented by the encoding of a base character followed by the encodings of one or more combining characters. For example, the **e with acute accent** graphical symbol can be represented in UCS-4 by (hex) 00 00 00 65 00 00 03 01 — the encoding for lower-case letter <e> followed by the encoding for a combining <acute accent>. This symbol can also be represented in UCS-4 by (hex) 00 00 00 E9 — the encoding for Latin small letter <e with acute accent>. A composite graphical symbol can thus have more than one encoding in UCS-4 (and also, similarly, in UCS-2). ISO / IEC 10646 defines three conformance levels:

1. Combining characters are not allowed.
2. Some combining characters are allowed for certain scripts, such as Arabic, Hebrew, Indic and Thai.
3. Combining characters are allowed with no restrictions.

The combinations of the three conformance levels with the two encoding forms gives six possible ways in which an implementation can support ISO / IEC 10646. Version R1.1 of the UNICODE standard is equivalent to just one of these ways: UCS-2 Level 3.

A degree of compatibility with ISO / IEC 646 is maintained, in that the characters encoded by ISO / IEC 646 are encoded by ISO / IEC 10646 using the ISO / IEC 646 codes preceded by the appropriate number of null octets (one in the UCS-2 form; three in the UCS-4 form). For example, upper-case **A** of the Latin alphabet is encoded as (hex) 41 by ISO / IEC 646, and as (hex) 00 41 by ISO / IEC 10646.

However, the ISO / IEC 646 encoding of any control or graphic character can appear as the leading octet of the encoding of a completely different character in the UCS-2 form of ISO / IEC 10646, or as any of the three leading octets of an encoding of the UCS-4 form. For example, the ISO / IEC 646 encoding of the End of Text (<ETX>) character appears as an octet of the ISO / IEC 10646 encodings of the characters of the Greek alphabet. This makes it hard to use the UCS-2 or UCS-4 encoding for data transmitted using communication protocols that assign special meanings to ISO / IEC 10646 control codes.

Recognising that this is a problem, ISO/IEC 10646 defines a UCS Transformation Format (UTF). When applied to an ISO/IEC 10646 encoding, this algorithm yields a 1, 2, 3 or 5 octet value that is guaranteed not to contain the ISO/IEC 646 encodings of any control character, or of the <SPACE> or <DEL> characters. Data encoded in accordance with ISO/IEC 10646, and then transformed by a UTF, can safely be transmitted using communication protocols that assign special meanings to ISO/IEC 646 control codes.

The algorithm defined by ISO/IEC 10646 (known as UTF-1) does not prevent encodings from containing the ISO/IEC 646 encoding of the slash character, (hex) 2F. This limits its use on POSIX-compliant systems, where the slash character is used to delimit segments of pathnames of files. (There are similar problems with many systems that are not POSIX-compliant.) A second UTF, FSS-UTF, was therefore defined by the X/Open-UniForum Joint Internationalization Group (JIG); it has been adopted by ISO as a normative annex to ISO/IEC 10646 under the name of UTF-8.[1] In this UTF, an octet with bit 8 set to zero can only appear as the single-octet representation of the identical ISO/IEC 646 encoding. As well as being safe for transmission by common communication protocols, such data can safely be processed by applications that handle file pathnames on POSIX-compliant systems.

Most current implementations use the UCS-2 form of encoding, because it is much more economical in its use of storage. A further transformation, known as UTF-16 (or ''shifted UNICODE'') has been defined to enable applications on such systems to use some of the characters that can be represented in UCS-4 but not in UCS-2. It does this by using pairs of UCS-2 code positions to represent UCS-4 characters.

The current practice for accommodating ISO/IEC 10646 and the UNICODE standard in open systems is to use the UTF-8 encoding for filestores and to define interface functions whose arguments are of type **wchar_t** (see Section 1.3) rather than of type **char**; these interface functions can if necessary parallel existing functions whose arguments are arrays of type **char**.

A full discussion of the issues pertaining to the use of ISO/IEC 10646 in open systems is contained in the X/Open **UCS** Technical Study.

1. See the referenced X/Open **UFT-8** Specification.

1.3 The C Programming Language

In internal machine storage, characters are held in bytes. A byte is a unit of machine storage containing at least 8 bits, each of which can take the value 0 or 1.

Often, the same encodings are used for characters held in machine storage as are used for characters in transmission.

The facilities of the programming language determine how characters held in machine storage can be manipulated by applications programs. For applications within the X/Open CAE, the most important programming language is C. The character handling facilities of the C programming language are of great importance with regard to the development of internationalised applications.

Early versions of the C programming language, such as that specified in **XPG1**, assumed a character encoding scheme similar to ASCII. They defined a **char** type such that a value of type **char** could be held in a single (8-bit) byte, and defined a character string to be an array of type **char** terminated by a null character. Many applications programs written using such versions of C use these facilities, and are not amenable to internationalisation, since they cannot handle multi-byte character set encodings.

In the version of C standardised by ANSI, and subsequently by ISO, some of the issues associated with internationalisation are addressed. The **char** type still has values that can be represented as single bytes, and character strings are still null-terminated arrays of type **char**. However, multi-byte character encodings are possible, and can be held in strings with several elements of type **char** representing each character. Also, the type **wchar_t** is provided for multi-byte character encodings. In ISO C it is defined to be such that its range of values can represent all codes for the largest supported character set.

A set of character and string handling functions that have arguments that are of type **wchar_t** and related types are defined in the MSE amendment to ISO C. For example, function *strcat*( ) has been used since the earliest days of C programming, but is unsuitable for use in internationalised programs because it has arguments of type **char ***. This constrains the language to be one that uses an 8-bit character set. Many languages use character sets that are not representable using 8 bits. The MSE amendment to ISO C includes the *wcscat*( ) function, which takes wide-character code arguments (type **wchar_t ***) and can be used in place of *strcat*( ) in internationalised programs.

Because strings are null-terminated, an encoding scheme used in conjunction with ISO C must not produce a null byte except as the encoding of the null character. The UCS-4 and UCS-2 encoding schemes do not have this property; therefore, use of the C language **char** data type as defined in ISO C in conjunction with the coded character set defined in ISO/IEC 10646 is problematic.

In addition to permitting flexibility of character sets and encodings, ISO C specifies a *locale* mechanism that can be used to enable application programs to be written without making assumptions about language and cultural conventions. ISO C specifies functions for handling characters, strings, date and time, and formatted input/output. The behaviour of these functions is affected by the current locale. This can be set by the applications program to reflect the language and cultural environment in which the application is executing. Application programs can also examine the current locale and modify their behaviour accordingly.

Character collation, classification and case conversion, and the format of numbers, monetary values and dates may all be affected by the locale. ISO C does not prescribe precisely how they are affected in any particular language and cultural environment (other than a basic default environment); it just specifies a general mechanism whose use is implementation-defined.

1.4 Internationalisation Support in POSIX

The locale mechanism of ISO C is extended by the POSIX.1 standard.[2] This provides a means whereby an application program can use a locale that has been established in its process environment. For example, this allows a system to be shipped with a repertoire of pre-defined locales. The user or system administrator selects the locales in which applications run. However, the POSIX.1 standard still specifies the general mechanism only, and contains no standardised descriptions of specific locales (other than the default locale).

Also, the POSIX.1 standard defines a Portable Filename Character Set, which it recommends for use in international applications. (It allows other characters to be used in filenames, but advises that such names are not portable between different language and cultural environments). This consists of the upper and lower-case characters of the Latin alphabet as used in English, the digits 0 to 9 and the period, underscore and hyphen characters (as found in ISO / IEC 646).

The interface specified in the POSIX.2 standard[3] provides for a system to support multiple locales and, optionally, to allow the user to define locales. The behaviour of the system utilities is affected by the currently established locale. For example, the *ls* utility lists files, sorted by name according to the collation sequence in the current locale.

The current locale also affects certain aspects of the command interpreter (*sh*), although the reserved words that have special meaning are all defined using a particular character set — the Portable Character Set — that is required to be present in every supported locale. This Portable Character Set is a superset of the Portable Filename Character Set defined in the POSIX.1 standard. It includes additional punctuation characters such as *{* and *}*.

Several of the utilities defined in the POSIX.2 standard can handle character-patterns called *regular expressions*. The meaning of ''regular expression'' is defined in terms of the current locale. For example, it is possible to specify the range of characters [a-z] as a regular expression; this would include the e-acute character in a French locale but not in an English one.

The definition of a locale includes the specification of an encoding of its characters. Stateless, but not stateful, multi-byte encodings are supported.[4]

---

2. The POSIX.1 standard is identical to IEEE Standard 1003.1. It specifies a programming interface to operating system services.
3. The POSIX.2 standard is identical to IEEE Standard 1003.2, which specifies a user interface to operating system services (commands and utilities).
4. A stateful encoding is one in which a code can set the interpreter into a state that affects the meaning of subsequent codes. An example of a stateful encoding is one that has a shift-lock code that causes subsequent codes for lower-case letters to be interpreted as the corresponding upper-case letters.

1.5 Internationalisation Support in the X/Open CAE

The need for internationalisation was stated in the first issue of the X/Open Portability Guide (**XPG1**). A trial-use definition of facilities to enable internationalised application programs to be developed was contained in the second issue (**XPG2**). Issue 3 (**XPG3**) included some mandatory facilities for the X/Open System Interface (XSI), which were largely aligned with the internationalisation facilities of the POSIX.1 standard and the ANSI C standard. They were expanded and refined in Issue 4 (**XPG4**) including full conformance with ISO C. (ISO C is based on, and technically equivalent to, the ANSI C standard.)

A more complete description of the development of internationalisation facilities can be found in the X/Open **Internationalisation** Guide. The differences between Issue 3 and Issue 4 of the XSI are summarised in the X/Open **Migration** Guide (Issue 4 is the latest version, published in July 1992.)

The XSH internationalisation facilities represent the most comprehensive, commonly agreed understanding of the requirement to date. They are summarised in Section 1.5.1.

Recent further work within the JIG has been concerned with internationalisation within a distributed systems environment. This concludes that the internationalisation facilities specified in the X/Open **XSH, Issue 4** Specification are not sufficient. It proposes further facilities and places an implicit requirement on the communication infrastructure. It represents the current direction of thinking and is summarised in Section 1.6.1 on page 8.

1.5.1 XPG4 Facilities

Firstly, the X/Open **XSH, Issue 4** Specification includes the **wchar_t** type of ISO C and the locale mechanism of the POSIX.1 standard.

Secondly, recognising that many of the traditional open systems facilities do constrain the language, culture or business environment assumed by the application, XSH includes a parallel Worldwide Portability Interface facility for each such traditional facility. These facilities are provided by the functions that are defined in the MSE amendment to ISO C.

While the X/Open **XSH, Issue 4** Specification includes both the traditional, non-internationalised, function definitions and the internationalised, Worldwide Portability function definitions, it recommends use of the latter for new developments, retaining the traditional definitions for compatibility with existing systems and applications.

1.6 Current Work

Work is continuing on the following topics.

1.6.1 Distributed Internationalisation Requirements

The JIG has produced the X/Open **DIS, Version 2** Snapshot. This document discusses the issues arising from the need for internationalised applications programs executing in a distributed environment. It considers multi-processor applications, where the program in each processor may be multi-threaded. In addition to applications that execute in a single locale (which may vary from user to user), it considers applications that, for a single user, process data created in multiple locales.

When distributed internationalised applications cooperate, it is important that they assume the same locale information. For example, if a list of names created on a system in Denmark is sorted into alphabetical order on a system in the USA, the American system must use the right collating rules (placing AA at the end of the list rather than at the beginning, for example). For this to be possible the following must be true:

- There must be a standardised means of describing locales.
- There must be a way of identifying particular locales.
- There must be a way of conveying locale information between communicating applications.
- It must be possible for an application to use the appropriate locale when processing information that has been created by another application.

The X/Open **DIS, Version 2** Snapshot describes the syntax and semantics of a naming scheme that identifies a locale across a heterogeneous network. Version 1 proposed that a registry of standard locales should be established. This registry has been set up by X/Open and is described in the X/Open **DIS, Version 2** Snapshot.

The X/Open **DIS, Version 2** Snapshot defines a set of functions that support multi-locale programs in a distributed environment. They are intended to be viewed as potentially being able to operate on code element strings of any data type, although they are specifically defined only for code element strings of types **char** and **wchar_t**. They include provision for single characters represented by multiple code string elements (for example, *e-acute* represented by two elements: *letter e* and *acute accent*). They are aligned with the functions defined in the MSE amendment to ISO C, and allow for *character set introducers*, *direction introducers* and possibly in future for other introducers; for example, to indicate locales (strings containing such introducers are sometimes referred to as *self-announcing data* or *tagged data*).

1.6.2 Definition and Registration of Locales

A registry of standard locales has been established by X/Open. The X/Open **Locale Registry Procedures** Guide describes how the registry operates. The locales in the registry can be obtained from X/Open. At the time of writing, the registry contains some 22 locales, including Danish, Dutch, English (American, British and Canadian), Faroese, French (Canadian), German (Austrian, German and Swiss), Greenlandic, Hungarian, Icelandic, Italian, Japanese, Latvian, Lithuanian, Polish, Portuguese and Romanian locales.

1.6.3 Complex Text Languages

The locale mechanism currently defined in **XPG4** covers the most commonly encountered differences between languages or cultural environments. However, it does not provide for all differences. In particular, it does not address the special needs of those languages that have been described as *complex text languages*. These can be defined as languages that have different layouts and forms of the text for presentation purposes and for processing purposes. These differences are generally concerned with:

- directionality; for example, in Arabic, Farsi, Urdu, Hebrew and Yiddish, the text flows mainly from right to left but includes segments that must be read from left to right
- shaping and composition of characters; for example, in Arabic, each character has a different form depending on whether it stands alone, is at the beginning of a word, is in the middle of a word, or is at the end of a word
- national numbers; for example, in Arabic, Thai, Chinese and Bengali, there are numeric characters other than the normal ''Arabic'' numerals (Arabic uses Hindi numerals), and the encodings of the ''Arabic'' numerals (hex 30-39 in ASCII) should be understood as representing these characters rather than the ''Arabic'' ones when the text of these languages is processed.

These issues are addressed in the X/Open **Layout Services** Snapshot.

1.6.4 Use of the UNICODE standard/ISO/IEC 10646

ISO/IEC 10646 represents a radical new direction in character set encoding standards. There are a number of questions relating to its use that are not yet settled. These include:

- Should all implementations support all characters defined in ISO/IEC 10646 (that is, treat them as valid input and perform valid comparisons on them), or should it be possible to define standard subsets so that an implementation need not support every character?
- Should UCS-dependent APIs (that is, APIs that assume that character data is encoded in accordance with ISO/IEC 10646 UCS-2) be defined?
- What are the appropriate scopes of use of the various UTFs that have been defined?

1.6.5 Testing of Internationalised Components

An internationalised system component should work in any language and cultural environment. This means that it must be tested in conjunction with a number of locales. The question of what locales should be used for testing purposes has been raised. It may be that new locales, incorporating particular combinations of characteristics, will be defined for testing purposes.

1.6.6 Distributed Internationalisation Framework

The X/Open **Distributed Internationalisation Framework** Snapshot sets the context for work on internationalisation in distributed systems. It provides an overview and analysis of the problem areas, but does not contain detailed interface specifications, which are in the X/Open **DIS, Version 2** Snapshot.

Chapter 2

The X/Open Common Desktop Environment

2.1 Introduction

The X/Open Common Desktop Environment (XCDE) specifications extend the X/Open Common Applications Environment (CAE) to enable applications programs to interact with people in a manner that is modelled on the way that desktops are used.

On a desktop, information relating to several activities can be kept readily accessible and can be used in carrying out the activity on which work is being done at any particular moment. Any information processing activity can be performed. There may be special provision, through diaries, memo pads, and so on, for performing common office tasks.

In an information processing system, the desktop is modelled by a user interface that has ''windows''. A user can have one or more windows for each of his activities. Each window contains a visible representation of information which is displayed to the user and which he may be able to create or modify. Changes in the information held by the system cause changes to the displayed representations, and changes made by users to the displayed representations cause changes to the information held by the system. A user can interact with any information processing application program in this way. There is special provision for applications such as diaries and message handling systems that support common office tasks.

This form of user interface is very powerful and flexible. From the point of view of the applications programmer, it requires a more sophisticated form of programming interface than a traditional character-based terminal interface. In particular, the application program does not directly control all aspects of the user interface. Rather, there are autonomous programs (''managers'') that handle many aspects of the interface (such as refreshing the display when the user selects a new activity) and whose operation can be influenced by the application program and by the user.

2.2 Elements of the XCDE

A user of the XCDE interacts with his system via the X Window System display and window handling system and the Motif style of user interface. He can customise some aspects of the user interface. The applications that he can use include an icon editor, a file manager, a mathematical calculator, a simple text editor, a calendar and appointments diary and an electronic mail system. A character-based terminal emulator enables him to use non-desktop applications. Printer management services enable him to print files and control printer operation. ''Help'' on how to use the system and its applications is available to him.

These services to the user are supported by the following elements of the XCDE:

- the X Window System display and window handling system, including the X Protocol handler, the X Library and the X Toolkit
- the Motif toolkit
- a manager that handles ''virtual screens'' called workspaces
- a manager that handles sessions of interaction between users and the system

- ‘‘front panel’’ facilities that allow certain aspects of the user interface to be defined
- a style manager that allows users to customise the system’s visual behaviour
- an icon editor application
- a file manager application
- a mathematical calculator application
- a simple text editor application
- a calendar and appointments diary application
- an electronic mail application
- a character-based terminal emulator
- a printer manager
- a ‘‘help’’ subsystem
- an inter-process messaging system (based on ‘‘ToolTalk’’) that enables applications to communicate with each other
- a ‘‘drag-and-drop’’ service that supports the transfer, under control of the user, of information between applications
- data-typing services that enable applications to categorise information
- execution management services that enable applications that have not been designed for the XCDE to be executed within the XCDE.

APIs to these components are available to the applications programmer. An application program can interact with the XDCE through a C-language interface or through a ‘‘scripting’’ interface that uses an extension of the shell command language.

Applications written by different people must cooperate within the XCDE. The XCDE specifications define a number of conventions and common information formats that enable applications to cooperate effectively.

The XCDE also contains services that enable applications programs to be created and integrated into the system, and services that enable applications and information files in the system to be associated with icons displayed at the user interface.

2.3 Internationalisation

Because the XCDE is so concerned with the presentation of information, internationalisation is a crucial aspect of its design.

In contrast to the traditional character-based model, the new ‘‘desktop’’ model of user interaction has a style that is in many respects quite independent of language and culture. For example, icons do not depend on language, and the action of pointing to an icon is an appropriate way of selecting an action, in any culture. But language and cultural differences have not been eliminated completely. Natural language text is still a part of the interface, and some icons have associations that are dependent on culture.

The requirement for international use has been taken into account in the design of the XCDE. There is provision for internationalisation of many of the remaining areas where the user interface is affected by language and culture. For example, there is provision for natural language text to be taken from message catalogues, such that different catalogues can be used

for different natural languages.

Nevertheless, there are still some problems for the application developer who wants to write a fully internationalised application in the XCDE. This study seeks to identify these problems, and to put forward recommendations for removing them.

Chapter 3

X Specifications

3.1 X Window System Protocol

3.1.1 Description

The X Protocol is the protocol that enables a computer on which an application is running (a *client*) to communicate with a computer that contains user interface hardware (a *server*) so that the application can interface to a user. It allows for the transfer of keyboard and other inputs from the server to the client, and of transfer from the client to the server of commands to display graphical shapes and text. It is described in the X/Open **X Window System Protocol** Specification.

3.1.2 Internationalisation Issues

3.1.2.1 Character Representations

Characters are represented in the protocol as indexes into font matrices. Indexes can be either 8 or 16 bits long. It would therefore not be possible to define a font for a character set of more than 65,536 characters. In particular, it would not be possible to define a font for the full ISO/IEC 10646 character set. This is unlikely to prove a practical limitation, however. If more than 65,536 of the possible ISO/IEC 10646 values are actually assigned to characters, it will be possible (and will probably be convenient) to use several fonts to display them.

3.1.2.2 Keyboard Input

The symbols on key caps are encoded by quantities called KEYSYMS. They are mapped to physical keys (encoded by KEYCODES) by a scheme that is heavily influenced by the traditional keyboard system of shift keys and other modifier keys. The KEYCODES are 8-bit integers, so a keyboard with more than 256 keys would present a problem. The KEYSYMS, however, are 32-bit integers (of which only 29 bits are actually used, half of them for vendor-defined codes). This scheme presents no serious immediate internationalisation issues but, taking a possibly speculative long-term view, there is a potential issue that should be raised. While the traditional Western keyboard has been adapted for use with languages and character sets of all kinds, it is by no means ideal for all of those languages. For example, for ideographic characters such as Chinese, a light pen and a small tablet that can recognise the characters might be easier to use. As technology develops, more efficient ways of obtaining input in non-Latin character sets will become practical. The KEYCODE/KEYSYM mechanism is unlikely to be able to cope with them without adaptation.

3.1.2.3 Defined KEYSYM Alphabets

The set of defined KEYSYMS includes the Latin-1, Latin-2, Latin-3, Latin-4, Kana, Arabic, Cyrillic, Greek, Technical, Special, APL and Hebrew characters plus a set of common keyboard symbols (RETURN, and so on). It is not clear how this standard set is to be extended to include other alphabets.

3.1.2.4 Error Strings

If connection set-up fails, the client receives a ‘‘reason’’ string or a set of information including a ‘‘vendor’’ string. It is likely that implementations would use these strings to convey textual messages. However, an internationalised application cannot interpret such a message (unless the strings are standardised) and cannot display it either, since it is not locale-dependent.

3.1.2.5 String Identifiers

Atoms have string names which by convention are restricted to the Latin-1 alphabet.

Font names are Latin-1 strings. They are matched by patterns that can include wildcards. Colour names and extension names are also Latin-1 strings.

3.1.2.6 Text Drawing

There are different sets of graphics primitives for drawing characters in fonts with 8-bit indexes and for drawing characters in fonts with 16-bit indexes. Programmers of internationalised applications must take care not to assume a particular index type for text that can be displayed in different character sets (and therefore in different fonts).

3.2 Xlib - C Language Binding

3.2.1 Description

Xlib, described in the X/Open **Xlib - C Language Binding** Specification, is a software library providing a C-language interface to the X Protocol described in the X/Open **X Window System Protocol** Specification.

3.2.2 Internationalisation Features

Xlib is designed to support variable-locale applications (but not multi-locale applications). However, an implementation of Xlib need not support all locales supported by its host. Function *setlocale*( ) can be used for locale announcement in an ANSI C environment, and an application can then call *XSupportsLocale*( ) to determine whether the current locale is supported by the implementation of Xlib.

The value of the current locale affects Xlib in:

- encoding and processing input method text
- encoding of resource files and values
- encoding and imaging of text strings
- encoding and decoding for inter-client text communication.

Parallel functions are defined using **char** and **wchar_t** data types for operations such as input handling and font handling.

Xlib provides for various different input methods to allow for text that uses complex characters to be entered.

Resource files are configuration files, intended to be set up by users and used to generate a resource database that can be queried by applications; for example, to determine window colours. Resource databases store properties. There is a locale bound to each resource database (see the description of *XrmLocaleOfDatabase*( ) in the X/Open **Xlib - C Language Binding** Specification).

3.2.3 Internationalisation Issues

3.2.3.1 String Identifiers

The encoding and interpretation of display names is implementation-dependent. Strings in the host portable character encoding are supported; support for other encodings is implementation-dependent. (The X Portable Character Set consists of the characters a to z, A to Z, 0 to 9, punctuation characters, space, tab and newline. The host portable character encoding is an encoding of the X Portable Character Set that is the same in all supported locales.) A particular format for display names is described for possible use on POSIX-conformant systems. It assumes availability of the period (‘.’) and colon (‘:’) characters.

The *DisplayString*( ), *XDisplayString*( ) and *XDisplayName*( ) functions return the display name as a string that presumably is null-terminated. This facility is said to be useful for printing error messages. The string is not locale-dependent.

Functions *ServerVendor*( ) and *XServerVendor*( ) return a null-terminated string describing the implementor, either in the host portable character encoding or in an implementation-defined encoding. It is not required to be locale-dependent.

Colours and other entities can be identified by character strings that are encoded in the host portable character encoding (or the result is implementation-dependent). String constants that make sense in English are defined for some identifiers.

Font names are passed across the API in null-terminated strings. Pattern matching can be applied to the names. Names and patterns are encoded using the host portable character encoding (or the result is implementation-dependent).

Font pathnames are encoded in a codeset that is dependent on the server implementation. This is a portability issue as well as an internationalisation issue. The application programmer or the system programmer must use the correct pathnames and the correct encodings for the servers to which his software must interface. It will be hard to develop an internationalised application in which the user can configure the font paths that are used.

It should be noted that names (of fonts, colours, hosts, and so on) should in general be encoded using the host portable character encoding on the client. This implies that the server should only use characters from the X Portable Character Set in such names. Presumably, they will be conveyed by the X Protocol in the encoding defined by ISO 8859-1, and may be converted to another encoding at the client, at the server, or at both. If such names are ever displayed to the user, an internationalised application will have to convert them (perhaps by table look-up) to a locale-dependent form.

3.2.3.2 Font Attributes

The built-in font property name CAP_HEIGHT is only useful in conjunction with character sets that distinguish upper and lower-case letters.

3.2.3.3 Text Directionality

Font structures include a horizontal directionality specification, but not a vertical directionality specification.

3.2.3.4 Error Strings

The *XGetErrorText*( ) and *XGetErrorDatabaseText*( ) functions return null-terminated strings in the encoding of the current locale. The use of null-terminated strings precludes encodings such as the UNICODE standard and ISO/IEC 10646 that contain embedded nulls.

3.2.3.5 Keyboard Input

Xlib relies on the KEYCODE/KEYSYM scheme defined in the X/Open **X Window System Protocol** Specification for keyboard input.

3.2.3.6 Simplified Keyboard Event Functions

The Latin-1 Keyboard Event Functions provide special (simplified) facilities for applications that assume the use of Latin-1 keyboards.

3.2.3.7 String Properties

Properties can be passed across the API as text strings, and there is provision for them to be encoded in accordance with the current locale. But the strings are presumably null-terminated.

3.2.3.8 Command Strings

The *XGetCommand*( ) and *XSetCommand*( ) functions get and set the command and arguments used to invoke a window's application. They must be encoded in the host portable character encoding, or the result is implementation-dependent.

3.2.3.9 Resource Files

Resource file syntax requires resource names to use Latin-1 alphanumeric characters plus the hyphen and underscore characters. Resource values can include any character except null or newline, and there are escape sequences that allow these to be included too. Also, there is a notation that allows any byte to be included. Resource databases are opaque; the entries are not manipulated directly but by using resource management functions. These take string arguments for resource names, which must be in host portable character encoding (or the result is implementation-dependent). Various retrieval functions return STRING-type resources as strings in the codeset of the current locale.

3.2.3.10 Cut Buffers

Cut buffers are properties containing text in STRING encoding. This restricts them to the Latin-1 alphabet. Internationalised applications can therefore not use this facility.

3.3 X Toolkit Intrinsics

3.3.1 Description

The X/Open **X Toolkit Intrinsics** Specification defines a widgets management C-language API.

A *widget* is instantiated by a window that enables a user to perform input or receive output or both. Each widget belongs to some *widget class*. The class that a widget belongs to largely determines the way that it behaves.

A widget class has associated with it a number of *actions* that widgets that belong to it can perform. They can be invoked in response to user-input events. This is done by *translation management*, which is table-driven. The *translation tables* can be extended and modified by application programmers.

As well as being invoked in response to user input, actions can be invoked directly by applications.

An instance of a widget has associated with it a number of stored information items called *resources* that it can use. Resources belong to *resource classes*. There is a mechanism for extending widget class records at run time.

Widgets, actions and resources are identified by character strings. This enables the application programmer to customise the behaviour of standard widgets to suit his application.

3.3.2 Internationalisation Features

The toolkit provides a mechanism that enables an application to set the locale.

The *XtResolvePathname*( ) function uses substitutions corresponding to XPG localisation conventions in order to search for a file.

An event filter mechanism for use by internationalised applications is defined for function *XtDispatchEvent*( ).

3.3.3 Internationalisation Issues

3.3.3.1 String Identifiers

Widget classes, actions, resource classes, resource representations and resources are identified by character strings. There are naming conventions that enable resource names and resource representation names to be derived from the names that the programmer gives to the source program data structure fields associated with them.

3.3.3.2 Default Font Resource

If no default font resource value is obtainable, the toolkit uses an implementation-dependent ISO 8859-1 font. This would not be appropriate in many locales. However, internationalised applications can always ensure that an appropriate font is available.

3.3.3.3 Error Strings

Error message strings may be constructed by the application and will be overridden by the contents of an external system-wide file, the *error database* file. It is not clear how these messages can be made locale-dependent. The name and path of the error database file is implementation-dependent, and so might be localised, but no satisfactory standard localisation method has yet been found.

3.3.3.4 Translation Table Syntax

The translation table syntax uses ISO Latin-1.

3.4 File Formats and Application Conventions

3.4.1 Introduction

The X/Open **File Formats and Application Conventions** Specification contains:

- Inter-Client Communications Conventions Manual (ICCCM)
- X Logical Font Description (XLFD)
- Compound Text
- Bitmap Distribution Format (BDF).

They are discussed in the following sections of this chapter.

3.4.2 Inter-Client Communications Conventions Manual (ICCCM)

3.4.2.1 Description

The Inter-Client Communications Conventions Manual (ICCCM) in the X/Open **File Formats and Application Conventions** Specification states the conventions that applications should follow in order to interwork with other applications. There are conventions in the following areas:

- selection mechanism
- cut buffers
- window manager
- session manager
- manipulation of shared resources
- device colour characterisation.

3.4.2.2 Internationalisation Issues

Atom Strings

Atoms are items of information identified by numbers. The information can include character strings. There is a convention that such strings should consist of Latin-1 alphabet characters encoded in accordance with ISO 8859-1.

Cut Buffers

When information is transferred between applications under user control by the *selection* method, the type of the information is indicated by an atom known as a *target atom*. There are two types of text that can be identified by the currently defined target atoms: STRINGs, consisting of Latin-1 characters, and COMPOUND TEXT, consisting of segments of characters with arbitrary encodings. When the *cut buffer* method of transferring information is used, only STRING text can be transferred. This means that the cut buffer mechanism cannot be used in many locales, and is thus not appropriate for use by internationalised applications.

String Properties

Some properties (items of information associated with windows) include text strings (for example, the WM_NAME property is an uninterpreted string that the client wants the window manager to display in association with a window). Properties are typed, and the type of a text property implies the encoding used for the text characters. The defined types for text property are STRING and COMPOUND TEXT (with the meanings given in **Cut Buffers** on page 22 for the atoms with these names). Internationalised applications should not use STRING properties to contain text for display.

Some properties (for example, the WM_CLASS property) contain more than one string. By convention, these are separated by nulls. Unless the null separators are interpreted as being of the same width as the widest character codes in the encoding of the text, this precludes the use of encodings such as the UNICODE standard or ISO/IEC 10646 that can contain embedded nulls.

3.4.3 X Logical Font Description (XLFD)

3.4.3.1 Description

The X Logical Font Description (XLFD) specification in the X/Open **File Formats and Application Conventions** Specification identifies those aspects of character renderings that can change from one font to another. It specifies how different fonts should be identified, and how their properties should be represented, in a desktop environment.

3.4.3.2 Internationalisation Issues

Font Attributes

Many of the font attributes are oriented towards Latin alphabets. For example, the CAP_HEIGHT property is meaningless for character sets in which there is no distinction between upper and lower-case letters. Also, such distinctions as the difference between Roman and Helvetica may not be meaningful in all alphabets.

String Identifiers

Font names are Latin-1 strings on which pattern matching can be done. They are case-insensitive. The name and pattern format must be supported by the X Protocol **OpenFont**, and so on, messages.

Character Set Registry

Character sets and encodings are registered in the X character set registry. The X character set registry registry appears partly to duplicate the functions of the X/Open Locale Registry, since programmers will need to relate these encodings to locales. Indeed, it would be better to name the locale rather than the encoding, since, for example, it may be necessary to sort words represented in the encoding.

Only one font — ISO 8859-1 — is guaranteed to be registered.

String Properties

FACE_NAME, COPYRIGHT and NOTICE are ‘‘human readable string’’ font properties (presumably Latin-1). FACE_NAME ‘‘may be used as feedback during font selection’’. There would be problems in displaying these strings in some locales. NOTICE gives copyright and trade mark information. Apart from the fact that the character set and encoding of this string will not be appropriate in all locales, there is also the problem that the form of the notice may be required to vary from one country to another. However, a discussion of international copyright law is beyond the scope of this study.

3.4.4 Compound Text

3.4.4.1 Description

The Compound Text specification in the X/Open **File Formats and Application Conventions** Specification defines how text that may include characters from multiple character sets should be encoded in a desktop environment.

3.4.4.2 Internationalisation Features

The encoding format is based on ISO 2022. There is support for standard encodings for the following character sets:

- ISO 8859-1 through ISO 8851-9 Latin
- Japanese 8-bit alphanumeric Katakana
- Chinese Hanzi
- Japanese Graphic
- Korean Graphic.

There is also support for arbitrary character set encodings. Their names should be registered with the X Consortium, and should when appropriate match the CharSet registry and Encoding registration described in the X Logical Font Description specification in the X/Open **File Formats and Application Conventions** Specification. These encodings can be single or multi-byte encodings, and can contain embedded nulls.

Text directionality can be indicated. Segments of text of different directionalities can be nested (to any nesting level).

A simple mapping to Latin-1 strings is defined to enable compound text strings to be used in resources.

3.4.4.3 Internationalisation Issues

Text Directionality

Horizontal, but not vertical, directionality can be indicated.

String Identifiers

Extension encoding names are Latin-1 strings.

Character Set Registry

The X/Open **File Formats and Application Conventions** Specification states that encodings should be registered with the X Consortium.

3.4.5 Bitmap Distribution Format (BDF)

3.4.5.1 Description

The Bitmap Distribution Format specification in the X/Open **File Formats and Application Conventions** Specification defines a standard format for representing character fonts.

3.4.5.2 Internationalisation Issues

Font Representations

Font representations are human-readable. They are expressed in an English-like formal language using the Latin-1 alphabet with U.S. ASCII encoding.

Chapter 4

XCDE Specifications

4.1 Motif Toolkit API

4.1.1 Description

The XCDE Motif toolkit enables the application programmer to give a particular ‘‘look and feel’’ to the user interface. It includes:

- the Motif window manager (**mwm**) command-line utility that handles the windows on the user’s desktop
- a number of widgets and associated functions
- functions associated with the User Interface Language (UIL)
- a number of other functions that perform operations related to the user interface (‘‘Toolkit Functions’’).

The Motif window manager, the above functions and associated data type definitions are specified in the X/Open **Motif Toolkit API** Specification. The X/Open **XCDE: Services and Applications** Specification defines the XCDE window manager, which consists of the Motif window manager plus some extensions.

4.1.2 Internationalisation Features

The LANG environment variable specifies the user’s choice of language for the **mwm** message catalogue, the **mwm** resource description file, and the search paths for resource description files and UID files. Window manager resource files can be used to give values for window manager resources. They are standard text files, for which an English-oriented syntax is specified, but which are in the language defined by LANG. This means that an appropriate window manager can be built for each locale.

In widget definitions, where appropriate, the encoding used for one resource is given by the value of another resource (for example, the **XmNiconNameEncoding** resource of the *TopLevelShell* widget class), or the directionality of the text in one resource is given by the value of another resource (for example, the **XmNstringDirection** resource of widget class *XmManager*).

4.1.3 Internationalisation Issues

4.1.3.1 String Identifiers

Various entities (for example, functions) are identified in resource files, in widget resources and in API function arguments using character strings. In function arguments, they are presumably null-terminated. In some cases, such as font names (XLFD strings, composed of Latin-1 characters, see the X/Open **File Formats and Application Conventions** Specification) and font tags (ISO/IEC 646 strings), their character sets and encodings are restricted. The specification of restricted character sets implies that the character set encoding of the locale must include the encoding defined in ISO 8859-1 as a subset.

4.1.3.2 Argument Lists

The **XmNargv** resource of the *Application Shell* widget class is a list of items of type **string**.

4.1.3.3 Accelerator Descriptions

Accelerators are described by **String** resources in a format similar to that used by the translation manager.

4.1.3.4 Uil String Formats

Source and object files are passed to *Uil*( ), the User Interface Language (UIL) compiler routine, as strings. Function *Uil*( ) passes to its error callback function a character string containing error message text, and a string consisting of the source line where the error occurred. The description of *Uil*( ) does not state that these strings use the character set encoding of the current locale.

4.1.3.5 Scale Widget Number Formats

XmScale widgets display numbers with decimal points. It is not clear whether the decimal separator is internationalised, and it is not clear whether national numerals are used in those locales that have them.

4.1.3.6 String Manipulation

String comparison and manipulation functions such as *XmStringHasSubstring*( ) are likely to produce counter-intuitive results in locales where glyphs can have alternative representations (for example, <e-acute> and <e> plus <acute accent>).

4.1.3.7 Text Directionality

Compound string functions such as *XmStringCreate*( ) do not allow for vertical directionality.

Text field functions implicitly assume horizontal, rather than vertical, directionality.

4.1.3.8 Text Widget Values

The **XmNValue** resource of widget classes *XmText* and *XmTextField* is of type **String** rather than **XmString**, and there is no resource to indicate the encoding used for the **XmNValue** resource. This appears to limit the degree to which these widgets can be used in internationalised applications. Moreover, the descriptions of these widgets do not state that their behaviour is entirely locale-dependent. However, from their descriptions, these widgets are clearly intended to be used in internationalised applications.

4.2 XCDE Definitions and Infrastructure

4.2.1 Introduction

The X/Open **XCDE: Definitions and Infrastructure** Specification contains an introduction, a glossary and descriptions of:

- XCDE data format naming
- the relationship of XCDE to the X Window System and Motif
- the C interface to some miscellaneous XCDE services
- the XCDE message services
- extensions to the Motif drag-and-drop services
- a C-language interface to data typing services
- a C-language interface to execution management services.

These descriptions are contained in a number of self-contained chapters of the X/Open **XCDE: Definitions and Infrastructure** Specification. Each of these chapters is discussed in a separate subsection below.

4.2.2 XCDE Data Format Naming

4.2.2.1 Description

The section on data format naming forms the entire contents of the chapter entitled ‘‘General Definitions and Requirements’’. It contains a description of the names that are used to identify data formats. Standard names are defined for a number of formats, such as encapsulated postscript, RFC 822 message format and the X PixMap format. A set of conventions is defined for the names of other data formats.

4.2.2.2 Internationalisation Issues

The names must be valid ICCCM selection target atoms, which implies that they use the Latin-1 alphabet defined in ISO 8859-1. According to the naming conventions stated in the X/Open **XCDE: Definitions and Infrastructure** Specification, they should be in upper-case with words separated by underscores.

4.2.3 X Window System and Motif

4.2.3.1 Description

The X/Open **XCDE: Definitions and Infrastructure** Specification contains a description of the relationship between the XCDE and the X Window System and Motif. It includes specifications of three additional widgets and of the C interfaces to some widget convenience functions for those widgets.

4.2.3.2 Internationalisation Implications

There are no internationalisation issues specifically associated with this section of the X/Open **XCDE: Definitions and Infrastructure** Specification.

4.2.4 Miscellaneous Desktop Services

4.2.4.1 Description

This chapter of the X/Open **XCDE: Definitions and Infrastructure** Specification contains C-language definitions of functions that initialise the desktop library and of a header file that includes a number of public constant definitions.

4.2.4.2 Internationalisation Issues

Applications, tool classes and library versions are identified in the API by character strings that presumably are null-terminated (see, for example, the *name* argument of *DtInitialize*( )).

4.2.5 Message Services

4.2.5.1 Description

The XCDE message services are based on the Sun Microsystems ToolTalk service. They support the creation, sending, reception and processing of messages between applications. Message reception is based on pattern matching criteria. The X/Open **XCDE: Definitions and Infrastructure** Specification contains:

- a specification of the C interface to message services
- specifications of command line interfaces to message-related utilities
- specifications of standard messages.

4.2.5.2 Internationalisation Features

The **tt_type_comp** utility is a form of compiler. The source code that it processes is contained in text files and, with certain flag settings (–h, –O, –p, –P, –v), it generates textual output. Also, some of the data items created by **tt_type_comp** include text strings. The X/Open **XCDE: Definitions and Infrastructure** Specification states that its execution is affected by internationalisation environment variables such as LANG. It is to be assumed that these determine the codeset and, where appropriate (for example, for help information), the language used for textual input and output, although the X/Open **XCDE: Definitions and Infrastructure** Specification does not state this explicitly. Other utilities (**ttcp**, **ttmv**, **ttrm**, **ttrmdir**, **ttsession**, **tttar**) are also affected by locale environment variables. It is similarly to be assumed that any textual output (such as help information) that they generate is in the language and codeset of the current locale.

The data type of the *contents* argument to the *Deposit* request message should be **string** unless the contents can include embedded nulls, in which case it must be **bytes**. The **bytes** data type allows application programmers to pass text encoded using an encoding that contains embedded nulls.

4.2.5.3 Internationalisation Issues

String Identifiers

A number of entities such as slots, files, processes and sessions, are identified by null-terminated character strings.

Message attributes and pattern values can include null-terminated arrays of type **char** and can include length-specified arrays of type **unsigned char**.

Message Trace Strings

The **ttsession** utility has a trace mode in which messages and their statuses are displayed. The *tt_message_print*( ) function returns a string containing a textual representation of a message trace, formatted as for the **ttsession** trace facility. It is to ''allow the application to dump out message that are received but not understood''. The *tt_pattern_print*( ) function returns a string containing a textual representation of a pattern, formatted in a similar way. There is no requirement for these textual representations to be locale-dependent.

Message status attributes have character strings associated with them (see the description of *tt_message_status_string*( ) in the X/Open **XCDE: Definitions and Infrastructure** Specification). The X/Open **XCDE: Definitions and Infrastructure** Specification states that the status string should be used by the application developer to amplify on, for example, why the application is failing a message. There is no requirement for these strings to be locale-dependent

The *tt_status_message*( ) function returns a null-terminated character string that describes a problem status code. There is no requirement for this string to be locale-dependent.

Locale Conflicts

A data item of type **otype** generated by **tt_type_comp** contains an optional command string that can be executed. A data item of type **ptype** can contain a *start* string that is executed by the shell to start a process. If the shell is internationalised (as it should be), these strings should be in the codeset of the current locale. This is the locale indicated by the environment of the background ToolTalk process **ttsession**, and could be the locale of the user or of the system administrator. However, **if tt_type_comp** is internationalised, they will be in the codeset of the application developer or, in some cases, of the application installer.

The X/Open **XCDE: Definitions and Infrastructure** Specification does not make it clear how this conflict should be resolved. It may be that, in practice, an internationalised application should use only command and start strings expressed in the character set and encoding of ISO 8859-1, and that users and system administrators should only use locales whose character set encodings include the encoding of ISO 8859-1 as a subset. Note that this would preclude encodings such as EBCDIC which have codes different from those of ISO 8859-1 for the Latin alphabet.

The *Get_Environment* request message reports the (string) value of a (string-named) environment variable. The name and the value are presumably both encoded in the codeset of the user's locale, established at run time. However, if the application developer supplies names or values for these strings (for example, as compiled string constants), they will be in the execution codeset of the compiler. The X/Open **XCDE: Definitions and Infrastructure** Specification does not state how this conflict should be resolved.

Similar considerations arise for other message arguments of string type, such as the string identifying the tool's current working directory returned by the *Get_Situation* request message. This is particularly the case for the *visual* argument of the *Get_XInfo* request message — an output argument for which a set of standard values (''StaticGray'', and so on) are defined; application programmers will naturally test this argument by string comparisons with compiled string constants.

4.2.6 Drag-and-drop

4.2.6.1 Description

The Motif drag-and-drop services are described in the X/Open **Motif Toolkit API** Specification. They allow for transfer of information between applications when the user ''drags'' an icon representing the information from one window to another, and ''drops'' it there. The X/Open **XCDE: Definitions and Infrastructure** Specification defines extensions, including convenience APIs, to the Motif drag-and-drop services.

4.2.6.2 Internationalisation Issues

String Identifiers

Some of the data items that can be passed to the drag-and-drop callback functions are character strings that presumably are null-terminated (for example, the file names in the *String* field of the **DtDndContext** structure). Some of these strings identify resources (see the description of *DtDndDropRegister*( ) in the X/Open **XCDE: Definitions and Infrastructure** Specification).

Drag-and-drop host names and file names are in STRING format, and are therefore restricted to the Latin-1 alphabet and ISO 8859-1 encoding.

Drag-and-drop buffer names are in STRING format, and are therefore restricted to the Latin-1 alphabet and ISO 8859-1 encoding.

Drag-and-drop Text

Textual data transferred using drag-and-drop services can be either Latin-1 alphabet text encoded according to ISO 8859-1 (STRING data) or compound text in arbitrary alphabets encoded using arbitrary encodings (COMPOUND_TEXT data).

4.2.7 Data Typing

4.2.7.1 Description

The XCDE data typing services provide capabilities that enable the attributes of a file or other data item to be determined from its name, in accordance with criteria held in a database. The X/Open **XCDE: Definitions and Infrastructure** Specification defines:

- a C-language API to XCDE data typing services
- the location and format of the files from which the database is created.

4.2.7.2 Internationalisation Features

A different database can be established for each locale.

In the files from which the database is constructed, file names are described by shell pattern-matching expressions as defined in the X/Open **XCU** Specification. These take account of internationalised regular expressions.

4.2.7.3 Internationalisation Issues

String Identifiers

Some of the API function arguments are character strings that presumably are null-terminated and that identify entities (for example, the *opt_name* argument of *DtDtsBufferToAttributeList*( )).

In the files from which the database is constructed, constructs for entity names must use the Host Portable Character encoding. Other constructs may use other codesets, but it is probable that codesets not compatible with ISO/IEC 646 would cause problems.

Boolean Strings

The *DtDtsIsTrue*( ) function tests a string for a Boolean value. The strings that it identifies as meaning ‘‘true’’ are all English words (or the number ‘‘1’’), and presumably are encoded in accordance with ISO 8859-1.

Configuration File Syntax

The syntax of the files from which the database is constructed is English-like.

String Manipulation

One of the data criteria sorting rules is that paths are ordered so that the longest is more specific. It is not clear whether, in codesets that include combining characters, combining characters count towards the length of a path name. Other rules may be affected in a similar way by codesets that contain combining characters.

4.2.8 Execution Management

4.2.8.1 Description

The XCDE execution management services support the handling of actions (applications and utilities that can be invoked from the desktop or by an application). Each system in the XCDE has access to an actions database (which may be distributed across several systems) that defines the relationship of actions to files associated with application programs and utilities. The X/Open **XCDE: Definitions and Infrastructure** Specification defines:

- a C-language API through which applications can load the database, query the database and invoke actions
- the **dtaction** utility that can be invoked from the command line and that in turn invokes an XCDE action
- the format of files from which the database is loaded.

4.2.8.2 Internationalisation Features

As for data typing services, a different database can be established for each locale.

For command actions, the database contains a string containing the command to be passed to the shell. It should be in the character set encoding of the user’s locale. This presents no problem, however, if there is a different action database for each locale.

4.2.8.3 Internationalisation Issues

Locale Dependence of Descriptions and Labels

The DESCRIPTION field of the actions database contains a textual description suitable for presentation to the user. There is no requirement that it should be locale-dependent. However the LABEL field, which contains a textual label for presentation to the user, is locale-dependent. This raises two questions:

- If there is a separate database for each locale, why does the LABEL field need to be locale-dependent?
- If there is some reason for the LABEL field to be locale-dependent, why does it not also apply to the DESCRIPTION field?

String Identifiers

Various API function arguments (for example, the *actionName* argument of *DtActionDescription*( )) contain character strings that identify resources and other entities. These entities are also identified by character strings in the **dtaction** command.

Command String

The command line text is returned, as a presumably null-terminated string, by *DtAction*( ).

4.3 XCDE Services and Applications

4.3.1 Introduction

The X/Open **XCDE: Services and Applications** Specification defines:

- APIs to the standard services and applications that are available to the user in the XCDE
- tools that are available to the programmer writing applications for the XCDE and to the system integrator building a system that includes the XCDE
- standard conventions that should be used by application designers so that the user interface will have the appropriate ''look and feel''.

These definitions are contained in a number of self-contained chapters of the X/Open **XCDE: Services and Applications** Specification. With some exceptions, each of these chapters is discussed in a separate subsection below. The exceptions are:

- the chapter on Window Management Services, which is discussed together with the Window Management part of the X/Open **Motif Toolkit API** Specification in Section 4.1
- the chapter on Calendar and Appointment Services, which is discussed together with the X/Open **XCS** Specification in Section 4.4.

4.3.2 Window Management Services

The XCDE window manager handles the windows on the user's ''desktop''. It is a superset of the X/Open Motif **mwm** window manager. It is discussed together with the Window Management part of the X/Open **Motif Toolkit API** Specification in Section 4.1.

4.3.3 Workspace Management Services

4.3.3.1 Description

The XCDE workspace manager provides support for multiple ''virtual screens'' known as *workspaces*. The X/Open **XCDE: Services and Applications** Specification defines a C-language API to the workspace manager. The API enables an application to query and control the state of a workspace, to add windows to workspaces, and to remove windows from workspaces.

4.3.3.2 Internationalisation Features

Workspace titles (a workspace title is displayed in the button for the workspace in the front panel) are passed across the API in character strings. Each such string is interpreted in the locale in which the workspace manager is running. This allows internationalised applications to use locale-dependent workspace titles.

4.3.3.3 Internationalisation Issues

The strings containing workspace titles are presumably null-terminated.

4.3.4 Session Management Services

4.3.4.1 Description

A session is the collection of applications, settings and resources that are present on a user's desktop from the time that he logs in to the time that he logs out. The XCDE session management services are described in the ICCCM sections of the X/Open **File Formats and Application Conventions** Specification. The X/Open **XCDE: Services and Applications** Specification defines a C-language API to them.

4.3.4.2 Internationalisation Features

An implementation of session management services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.4.3 Internationalisation issues

Filenames and pathnames are passed across the API as character strings, that presumably are null-terminated (see, for example, the description of *DtSessionRestorePath*( ) in the X/Open **XCDE: Services and Applications** Specification).

4.3.5 Help Services

4.3.5.1 Description

The XCDE help services provide information to the user about the system, the desktop environment and the applications that he can use. They are provided to the user through a number of widgets on the desktop. The X/Open **XCDE: Services and Applications** Specification defines the widget classes of these widgets.

4.3.5.2 Internationalisation Features

The help system is internationalised in that it enables a different set of help text to be provided for each locale. It does not provide for that text to follow a common format and cross-reference structure; the imposition of a common structure for help text in different locales is left to the application programmer. (See the *HelpTag* format description in the X/Open **XCDE: Services and Applications** Specification.)

For resources containing file pathnames, there is a mechanism for making the pathname dependent on the LANG environment variable. This enables an internationalised application to use an appropriate set of files for each locale.

There is provision for an application to deliver localised help into a non-localised XCDE environment (see the description of *DtHelpSetCatalogName*( ) in the X/Open **XCDE: Services and Applications** Specification).

4.3.5.3 Internationalisation Issues

Dialog widgets and cursors are identified in the API by character strings that presumably are null-terminated (see the *DtCreateHelpDialog*( ) and *DtHelpReturnSelectedWidgetId*( ) function descriptions in the X/Open **XCDE: Services and Applications** Specification).

4.3.6 Calendar and Appointment Services

The chapter on Calendar and Appointment Services is discussed together with the X/Open **XCS** Specification in Section 4.4.

4.3.7 Mail Services

4.3.7.1 Description

The XCDE mail services allow the user to send, receive and manipulate electronic mail messages. The message format defined in the RFC 822 Internet Specification is assumed, and the messages can be MIME-encoded and have attachments, as described in the RFCs 1521 and 1522 Internet Specifications.

4.3.7.2 Internationalisation Features

An implementation of XCDE mail services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.7.3 Internationalisation Issues

Mail Message Header Fields

The RFC 822 Internet Specification requires use of ASCII text for message headers and text, but not (if MIME is used) for attachments. This means that a user cannot in general use his national character set for message addresses, subjects, and so on. This applies for national as well as for international messages.

Locale Conflicts

A user can display lists giving message senders, subjects, and so on. These, if in ASCII, could be in a different locale from the current session locale. They will still usually be displayable, since ASCII is effectively a subset of most character sets used in computer systems.

Mail Message Text

A user whose locale does not have ASCII as its character encoding must make the text of his message an attachment rather than part of the message proper, which is an inconvenience to him. For example, the mail system is required to support the capability of displaying the text of a message. This is of little use for non-ASCII messages; in this case the user would prefer to display the first attachment.

Mail Aliases

A user must be able to maintain a list of personal mail aliases. For internationalised operation, he should be able to define aliases using the character set and encoding of his locale. No requirement to allow this is stated. (Normally, it would be taken for granted that a user could use the character set and encoding of his locale. However, as the names of message addressees must be in ASCII, it is important to state explicitly that aliases can be in other character set encodings.)

4.3.8 File Management Services

4.3.8.1 Description

The XCDE file management services provide a user interface for manipulation of objects (including files) and folders, and for application execution. The user accesses these services by invoking actions in desktop widgets. The X/Open **XCDE: Services and Applications** Specification defines these actions.

4.3.8.2 Internationalisation Features

An implementation of XCDE file management services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation. This allows a user to manipulate files that have been named in his own locale. It does not necessarily enable him to manipulate files that have been named in other locales.

4.3.9 Front Panel Services

4.3.9.1 Description

A front panel appears in every XCDE workspace. It is a window that enables many of the basic parameters of the desktop to be customised. The X/Open **XCDE: Services and Applications** Specification specifies the front panel capabilities that must be provided by an implementation of the XCDE, and defines the format of the configuration files that give the values of the front panel parameters in installed systems.

4.3.9.2 Internationalisation Features

An implementation of XCDE front panel services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.9.3 Internationalisation Issues

The configuration files have an English-like syntax. The character set and encoding used for the configuration files is not specified. It is likely that many implementations will assume ASCII.

4.3.10 Text Editing Services

4.3.10.1 Description

The XCDE text editing services enable the user to create and edit short documents, using the **DtEditor** widget. The X/Open **XCDE: Services and Applications** Specification defines:

- the **DtEditor** widget
- a C-language API to the XDCE text editing services
- the **dtpad** utility that can be invoked from a command line
- text editing actions that can be invoked from widgets
- inter-application (ToolTalk) messages supported by XCDE text editing services.

4.3.10.2 Internationalisation Features

An implementation of XCDE text editing services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

The **DtEditor** widget supports locales with single and multi-byte character set encodings (but presumably does not support character set encodings such as the UNICODE standard and ISO/IEC 10646 that contain embedded nulls).

The **dtpad** editor utility is affected by localisation of environment variables such as LANG.

The XCDE text editing services support the Tooltalk C_STRING message for text in an arbitrary codeset.

In a similar way to Motif text widgets, the **DtEditor** widget supports horizontal (left-to-right and/or right-to-left) directionality of text.

Localisation resources are defined that allow localisation of dialogues. This approach seems an excellent one from the point of view of internationalisation. It could usefully be followed in other widgets, such as the Motif text widgets.

The text that is edited can be passed over the API in null-terminated strings, in strings of type **wchar_t** or in sized buffers (see, for example, the description of *DtEditorAppend*()). This should be sufficient for a fully-internationalised application, and even allows for character set encodings that contain embedded nulls.

4.3.10.3 Internationalisation Issues

String Identifiers

Editor widgets are identified in the API by character strings that presumably are null-terminated (see the *DtCreateEditor*() function description in the X/Open **XCDE: Services and Applications** Specification).

Text Directionality

The **DtEditor** widget does not support vertical text directionality.

String Manipulation

Certain functions (*DtEditorChange*(), *DtEditorFind*() and *DtEditorInvokeFindChangeDialog*()) perform string manipulation and comparison operations such as search-and-replace. These operations are locale-dependent. However, the X/Open **XCDE: Services and Applications** Specification does not make it clear what happens when a locale allows a character to be encoded in two or more different ways (such as when an accented character can be encoded either as a single character or as a character with a combining accent character). It is likely that some string comparison operations will not work in the way that the user expects when a character can be encoded in more than one way.

Pasted Segment Directionality

Variable horizontal directionality is supported. However, the description of *DtEditorPasteFromClipboard*( ) does not make it clear whether, when a segment is pasted into a segment with reversed directionality, the directionality of the pasted text is reversed, or the pasted text becomes a nested segment with its own directionality.

4.3.11 Icon Editing Services

4.3.11.1 Description

The XCDE icon editing services enable users to create and modify icons. The X/Open **XCDE: Services and Applications** Specification defines the actions that the user can invoke from widgets in order to do this.

4.3.11.2 Internationalisation Features

An implementation of XCDE icon editing services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.12 GUI Scripting Services

4.3.12.1 Description

The XCDE GUI scripting services provide extensions to the shell programming language defined in the X/Open **XCU** Specification that allow access to the XCDE services. The X/Open **XCDE: Services and Applications** Specification defines the **dtksh** utility that can be invoked from the command line and that provides these services.

4.3.12.2 Internationalisation Issues

The X/Open **XCDE: Services and Applications** Specification states that the **dtksh** utility supports fully localised shell scripts, and it describes how a localised shell script is created. However, more detail could be given on the meaning and implications of localisation. For example:

- When **dtksh** converts string values to ‘‘an appropriate internal representation’’, is this dependent on the current locale?
- How is multi-character set compound text represented in a single-language locale?

4.3.13 Terminal Emulation Services

4.3.13.1 Description

The XCDE terminal emulation services provide a window for applications written for character-based terminals. The services are available in two forms: a stand-alone client and a widget. The terminal emulation provided is partly based on the VT220 terminal, and is compatible with ANSI X3.64 standard and ISO/IEC 6429. The X/Open **XCDE: Services and Applications** Specification defines:

- the terminal emulation capabilities that an XCDE implementation must support
- a C-language API to XCDE terminal emulation services

- the **DtTerm** terminal emulation widget
- the **dtterm** terminal emulation utility that can be invoked from the command line
- the actions that can be invoked to provide terminal emulation services
- the format of the data stream (including escape sequences) recognised by the terminal emulation.

4.3.13.2 Internationalisation Features

An implementation of XCDE terminal emulation services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

The behaviour of the **dtterm** utility is dependent on internationalisation environment variables such as LANG.

4.3.13.3 Internationalisation Issues

It is not clear how far languages that do not use the ASCII character set, and complex text languages in particular, can be supported with the defined terminal emulation. For example, in insert mode, characters are stated to be moved ‘‘to the right’’.

4.3.14 Style Management Services

4.3.14.1 Description

The XCDE style management services enable users to customise the visual elements and system behaviour of the XCDE. The X/Open **XCDE: Services and Applications** Specification defines the capabilities that an implementation of XCDE style management services must provide, and the style management actions that can be invoked.

4.3.14.2 Internationalisation Features

An implementation of XCDE style management services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.15 Application Building Services

4.3.15.1 Description

The XCDE application building services provide the application developer with aid in assembling graphical objects into the user interface and with generation of appropriate calls to XCDE API routines. The X/Open **XCDE: Services and Applications** Specification defines:

- the capabilities that an implementation of the XCDE application building services must support
- the **dtcodegen** utility that can be invoked from the command line and that provides XDCE application building services
- the actions that can be invoked to provide XDCE application building services.

4.3.15.2 Internationalisation Features

An implementation of XCDE application building services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

The behaviour of **dtcodegen** is affected by internationalisation environment variables such as LANG.

The developer can cause the code generator to determine whether internationalisation message-handling functions (*catgets*( ), and so on) should be used in the application, and whether a message source text file should be generated.

4.3.15.3 Internationalisation Issues

Source Code of dtcodegen

It is not clear whether the source codeset of **dtcodegen** programs is locale-dependent.

Object Palette Limitations

The object palette includes standard Text Pane and Text Field objects, but does not include standard date, time, number or money objects. This means that the application developer does not have to be concerned with handling text in an internationalised application, but he does have to be concerned with handling other data whose format is locale-dependent.

Building Help Text

It is not clear whether message catalogue source file generation applies to help text.

4.3.16 Application Integration Services

4.3.16.1 Description

The XCDE application integration services enable application developers and system administrators to integrate applications into the XCDE. The X/Open **XCDE: Services and Applications** Specification defines:

- the **dtappintegrate** utility that can be invoked from the command line and that provides XCDE application integration services
- the application integration actions that can be invoked.

4.3.16.2 Internationalisation Features

The **dtappintegrate** utility allows for files in selected or all locales to be integrated.

4.3.17 Action Creation Services

4.3.17.1 Description

Actions provide the ability to associate an application with an icon on the desktop. Data types provide the ability to associate a data file with an icon on the desktop. The XCDE action creation services enable users to define actions and data types. They are provided through a set of predefined actions that are specified in the X/Open **XCDE: Services and Applications** Specification. The X/Open **XCDE: Services and Applications** Specification also defines the capabilities that an implementation of XCDE action creation services must support.

4.3.17.2 Internationalisation Features

An implementation of XCDE action creation services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.17.3 Internationalisation Issues

Action creation services are required to support the association of help information with action icons. There is, however, no requirement for the help information to be localisable. In a multinational organisation, a user or system administrator defining an action for system-wide use might wish to define help information for use in more than one locale.

4.3.18 Print Job Services

4.3.18.1 Description

The XCDE print job services provide information to users about printers and print jobs. The X/Open **XCDE: Services and Applications** Specification defines the capabilities that an implementation of the XCDE print job services must support, and defines the print job service actions that can be invoked.

4.3.18.2 Internationalisation Features

An implementation of XCDE print job services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.19 Calculator Services

4.3.19.1 Description

The XCDE calculator services provide basic computation capabilities to users. The X/Open **XCDE: Services and Applications** Specification defines the capabilities that an implementation of the XCDE calculator services must support, and defines the calculator service actions that can be invoked.

4.3.19.2 Internationalisation Features

An implementation of XCDE calculator services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

4.3.19.3 Internationalisation Issues

It is not entirely clear what it means for an implementation of calculator services to be internationalised. For example:

- Does it mean that the number separators and decimal characters are locale-dependent (so that 10001/8 appears as 1,125.125 in an English locale and as 1.125,125 in a French one)?
- In locales that have special (non-Arabic) character sets, in which character set are numbers displayed?

4.3.20 Application Conventions

4.3.20.1 Description

The ''Application Conventions'' chapter of the XCDE describes font and icon conventions that applications should follow.

4.3.20.2 Internationalisation Features

There are requirements for fonts and font names for use in locales that include the Latin-1 character set (as defined in ISO 8859-1). There are guidelines, but not requirements, for other locales.

4.3.21 Application Style Checklist

4.3.21.1 Description

The ''Application Style Checklist'' chapter of the XCDE states the style requirements for XCDE applications.

4.3.21.2 Internationalisation Features

Although it does not define in detail the variations required for non-English locales, the X/Open **XCDE: Services and Applications** Specification does indicate the areas where such variations will apply, and it is generally clear what the variations should be.

4.3.21.3 Internationalisation Issues

Command and Filename Encodings

When the user selects files or commands, it is possible that the names of some of the files and commands that are available to him do not use the character set of the user's locale. There is no guidance as to how such names should be displayed.

Operating System Messages

The X/Open **XCDE: Services and Applications** Specification recommends that messages from the underlying operating system should be ‘‘translated’’ into non-technical terms before being given to the user. Presumably, this should include localisation. However, this will not be practical if the operating system generates non-localised text messages. The X/Open **XCDE: Services and Applications** Specification also states that the application optionally should not rely on error messages from the operating system. It might be better to state categorically that an internationalised application shall not display to the user non-localised text generated by the underlying operating system.

The X/Open **XCDE: Services and Applications** Specification states that the application optionally should write error messages to the XCDE error log when it is not appropriate to display them to the user, but when they may nevertheless be useful in diagnosing problems. For messages generated by the application, it might be more appropriate to log error codes and to have a utility that converts these codes to error messages in the locale of the system administrator or user who is using the error log.

4.4 Calendaring and Scheduling API

4.4.1 Description

The XCDE calendar and appointment services enable users to maintain appointment diaries, and to browse and update their own and other users' diaries in order to set up group meetings.

A C-language API to the XCDE calendar and appointment services is defined in the X/Open **XCDE: Services and Applications** Specification and the X/Open **XCS** Specification. The X/Open **XCDE: Services and Applications** Specification also defines:

- the capabilities that an implementation of XCDE calendar and appointment services must support
- utilities that can be invoked from the command line and that provide calendar and appointment services
- actions that can be invoked from the desktop or by applications and that provide calendar and appointment services
- inter-application (ToolTalk) messages used by calendar and appointment services
- the data formats of calendar archive files and calendar entries.

4.4.2 Internationalisation Features

The X/Open **XCDE: Services and Applications** Specification states that an implementation of calendar and appointment services is required to be internationalised in accordance with the X/Open **XSH** Specification, the X/Open **Xlib - C Language Binding** Specification and the X/Open **Motif Toolkit API** Specification, and must support any locales supported by the associated Xlib implementation.

The X/Open **XCS** Specification states that every calendar has a character set (there is an implementation-specific default). It defines the CSA_CAL_ATTR_CHARACTER_SET attribute, support for which is optional. It defines values IBM 437, IBM 850, Microsoft 1252, Apple ISTRING, UNICODE standard, TSS T61, TSS IA5, ISO/IEC 10646, ISO/IEC 646 and ISO 8859-1 for this attribute and allows implementations to support others. The calendar character set is presumably set through the *calendar_attributes* argument to *csa_add_calendar*( ).

The X/Open **XCS** Specification states that a session character set is identified by the *character_set* argument of *csa_logon*( ). Its values are implementation-specific.

4.4.3 Internationalisation Issues

4.4.3.1 Locale Conflicts

It is not clear whether the session character set established by *csa_logon*( ) can be different from the calendar character set. If it can be different, it is not clear how the implementation behaves when the session character set is different from the diary character set. For example, it might perform character conversion, in which case there should be some indication if information is lost.

The X/Open **XCDE: Services and Applications** Specification states that the behaviour of the utilities depends on the internationalisation environment variables (LANG, and so on). As with the session character set, it is not clear whether the character set and encoding of the locale can be different from the character set and encoding of the calendar, or how the implementation should behave if they are different.

The X/Open **XCDE: Services and Applications** Specification states that a user can change his calendar display to that of another user. It is not clear what happens if the two users work in different locales.

The X/Open **XCS** Specification states that a natural language can be specified by a combination of the (optional) CSA_CAL_ATTR_COUNTRY and CSA_CAL_ATTR_LANGUAGE attributes. They contain ISO 3166 and ISO 639 values respectively. It is not clear how the implementation behaves if this language is different from that defined by the currently established locale.

The X/Open **XCS** Specification defines function *csa_update_calendar_attributes*( ) which updates the calendar attribute values for a calendar. It is not clear what happens if an attempt is made to change the calendar codeset using this function. In particular, the X/Open **XCS** Specification does not state whether all of the entries should be re-encoded.

The X/Open **XCS** Specification defines function *csa_add_entry*( ) which adds an entry to a calendar. The entry can have attributes. It is not clear whether they can include a codeset attribute. If they can:

- It is not clear how the implementation behaves if this is different from the session codeset and/or the calendar codeset.
- It is not clear how the implementation behaves if an attempt is made to change this codeset through the *csa_update_entry_attributes*( ) function — are character values re-encoded?

4.4.3.2 Date, Time and Number Formats

The X/Open **XCDE: Services and Applications** Specification states that dates and times can be specified to some of the utilities (for example, via the **−d** and **−s** flags of **dtcm_insert**) and defines a particular form in which they are to be represented. This form may not be appropriate in all locales (some locales use *dd/mm/yy* rather than *mm/dd/yy*, for example). Also, it is not clear whether the numbers can be non-arabic numerals in locales that include non-arabic numerals.

4.4.3.3 Calendar Archive Names and Values

The X/Open **XCDE: Services and Applications** Specification states that, in the calendar archive file format, names that are ISO 8859-1 strings and values can contain null-terminated strings, interpreted relative to the character set for the entry containing them. However, it is not clear where this character set is identified in the entry description.

4.4.3.4 String Manipulation

The X/Open **XCDE: Services and Applications** Specification states that appointments can be searched for using string searches. It is not clear how these searches operate in locales where there are different representations for a single character (for example, <e-acute> and <e> plus <acute accent>).

4.4.3.5 Encoding of csa_logon( ) Arguments

It is not clear what character set and encoding is used for the arguments of *csa_logon*( ). The character set and encoding established by the *character_set* argument should apply to the *user*, *password*, *required_csa_version* and *logon_extensions* arguments. However, it cannot apply to the *character_set* argument itself. The most likely explanation is that the *character_set* argument must use Latin-1 characters encoded in accordance with ISO 8859-1, but the X/Open **XCS** Specification does not state this.

4.4.3.6 Rule Syntax

In the X/Open **XCS** Specification, exception rules and recurrence rules are specified as strings with an English-oriented formal grammar.

Chapter 5

Summary and Recommendations

5.1 Introduction

This section of the report contains a summary of the internationalisation issues and recommendations for resolving them. Provisional versions of these recommendations have been discussed by the X/Open Desktop Working Group and the Joint Internationalization Group of X/Open and UniForum. The recommendations as stated here embody the conclusions of those groups.

5.2 Character Representations

There is a practical limitation of 65,536 on the number of characters in a font (see Section 3.1.2.1 on page 15).

There are a number of alphabets that have more than 65,535 characters. They are usually dealt with by allocating multiple fonts to them. For example, three fonts are needed to represent Japanese characters, and three fonts are needed to represent Chinese characters. This is satisfactory, but guidelines are needed on how multiple fonts are to be used for a single character set.

There are some guidelines in X11R6, which provides for the **/usr/lib/X11/locale/%L/XLC_LOCALE** file to define the fonts for a single codeset that consists of multiple character sets. This aspect of X11R6 should be incorporated into the XCDE (which is currently based on the previous release, R5, of X11).

5.3 Font Attributes

Some font attributes are oriented towards Latin fonts (see Section 3.2.3.2 on page 18 and **Font Attributes** on page 23).

The Complex Text Languages requirements are relevant to this. They are discussed in the X/Open **Layout Services** Snapshot.

There are probably of the order of tens of attributes that should be investigated, and that may lead to the definition of further attributes to meet the needs of users of non-Latin alphabets.

It is likely that the X implementors will be the best people to define new attributes. Arrangements should be made so that if X/Open identifies a need for attributes that are not defined in the XCDE, then this need can be fed into the X implementors as a requirement.

5.4 Text Directionality

The XCDE supports text with varying horizontal directionality (right-to-left, left-to-right) but does not support text with vertical directionality (top-to-bottom or bottom-to-top). See Section 3.2.3.3 on page 18, **Text Directionality** on page 24, Section 4.1.3.7 on page 28 and **Text Directionality** on page 39.

Vertical directionality will probably be a requirement at some time in the future. This is made clear in the X/Open **Layout Services** Snapshot. It should be considered as a possible requirement for future issues of the XCDE specifications.

5.5 Pasted Segment Directionality

The description of *DtEditorPasteFromClipboard*( ) does not make it clear how segment pasting works in text with reversed directionality (see **Pasted Segment Directionality** on page 40).

Ideally, the directionality of the pasted segment should be context-sensitive. This is difficult to specify, however.

The ICCCM states that the destination application supplies a property and the originating application converts the information to the type of that property (or, if it cannot perform the conversion, the operation fails). Properties of type COMPOUND_TEXT should be specified by the destination application to cause the originating application to provide directionality information. If the source and destination applications can use the same services for handling shaping and re-ordering (such services are described in the X/Open **Layout Services** Snapshot), then all aspects of shaping and directionality can be preserved.

The XCDE specifications should be left unchanged.

5.6 Keyboard Input

The keyboard input recognition mechanism does not look to be appropriate for possible future new input methods (see Section 3.1.2.2 on page 15 and Section 3.2.3.5 on page 18).

This issue is not of immediate concern, and no action is proposed in respect of it.

5.7 Defined KEYSYM Alphabets

The set of defined KEYSYMS includes the characters of a number of alphabets (see Section 3.1.2.3 on page 15). As the XCDE is used more widely in locales with other alphabets, it will be appropriate to extend the set of defined KEYSYMS.

The X Consortium is the body that can best perform this task.

5.8 Simplified Keyboard Event Functions

The Latin-1 Keyboard Event Functions provide special (simplified) facilities for applications that assume the use of Latin-1 keyboards (see Section 3.2.3.6 on page 18). This is a convenience for programmers writing non-internationalised applications in locales that use the Latin-1 character set, but it does not affect the writing of internationalised applications.

The XCDE specifications should be amended to state that internationalised applications should not use these functions.

5.9 String Properties

Some properties contain Latin-1 text strings (see **String Properties** on page 23 and **String Properties** on page 24).

The requirement not to break implementations is overriding, and existing property specifications should not be changed. The specifications should warn that such text must not be displayed by internationalised applications. No new such properties should be defined.

5.10 Error Strings

Non-localised text conveying error or status descriptions can be conveyed by the X Protocol (see Section 3.1.2.4 on page 16) and returned by some API functions (see Section 3.3.3.3 on page 20, **Message Trace Strings** on page 31 and **Operating System Messages** on page 45).

These strings should not be internationalised but, where non-localised strings are returned, the specifications should warn that internationalised applications must not display the text to users. It should be noted that applications can define message catalogues containing strings that can be used in place of those conveyed by the protocol.

5.11 Null-terminated Strings

Strings passed to or returned by functions or contained in ToolTalk messages are often null-terminated (see Section 3.2.3.1 on page 17, Section 3.2.3.4 on page 18, Section 3.2.3.7 on page 18, Section 4.1.3.1 on page 27, Section 4.2.4.2 on page 30, **String Identifiers** on page 30, **String Identifiers** on page 32, **String Identifiers** on page 33, **Command String** on page 34, Section 4.3.4.3 on page 36, Section 4.3.5.3 on page 36, **String Identifiers** on page 39 and Section 4.4.3.3 on page 47). Lists of strings in properties are null-separated (see **String Properties** on page 23).

The use of null-terminated or null-separated strings presents problems when encodings such as the UNICODE standard and ISO / IEC 10646 that contain embedded nulls are in use.

To overcome these problems, functions should ideally treat strings as being multi-byte encoded and terminated by a number of null bytes that is appropriate for the encoding. However, the identity of the encoding need not always be available to XCDE functions that process character strings. Where applications use only the Host Portable Character Set, no problems will arise; in many cases the function descriptions warn that use of other characters may lead to implementation-dependent results, but for some functions (for example, *DisplayString*( ), *XDisplayString*( ) and *XDisplayName*( )) this caveat is missing.

The XCDE specifications should be amended so that a caveat statement appears in the descriptions of all functions to which it is relevant.

For ToolTalk messages, the ToolTalk protocol makes it clear which encoding is used.[5]

5.12 String Identifiers

Various entities including hosts, displays, fonts, widget classes and resources are identified in atoms, in ToolTalk messages, in program source code and in configuration files by character strings, often restricted to the Latin-1 alphabet. (See Section 3.1.2.5 on page 16, Section 3.2.3.1 on page 17, Section 3.3.3.1 on page 20, **Atom Strings** on page 22, **String Identifiers** on page 23, **String Identifiers** on page 25, Section 4.1.3.1 on page 27, Section 4.2.2 on page 29, Section 4.2.4.2 on page 30, **String Identifiers** on page 30, **String Identifiers** on page 32, **String Identifiers** on page 33, **String Identifiers** on page 34, Section 4.3.3.3 on page 35, Section 4.3.4.3 on page 36, Section 4.3.5.3 on page 36, **String Identifiers** on page 39 and Section 4.4.3.3 on page 47.)

The use of such identifiers in source code affects system programmers and application programmers but does not affect users.

The use of such identifiers in configuration files could affect users, in particular system administrators, as well as programmers. Configuration files are used to create databases such as the resource database, the data types database and the actions database. (See Section 3.2.3.9 on page 19.) In general, it is possible to define a set of configuration files in any locale, and to make the pathname which is used to access the configuration files when the database is built locale-dependent. However, while a value in the databases can use any character set and encoding, identifiers are generally required to use Latin-1 characters encoded in accordance with ISO 8859-1. This effectively restricts the supported locales to those whose character set encoding is an extension of ISO 8859-1. Not all encodings have this property. Those that follow ISO 2022 do but others, such as the UNICODE standard, do not.

Allowing string identifiers to use arbitrary character sets and encodings could present technical difficulties (see Section 3.1.2.5 on page 16).

A restriction to characters of the X Portable Character Set is acceptable; a restriction to the characters of the Latin-1 alphabet is not. There is an overriding requirement not to change the X Protocol (for compatibility and interoperability reasons), so references in the X Protocol specification to ISO 8859-1 should not be changed. However, this does not apply to Xlib and the other XCDE specifications.

The XCDE specifications should be amended as follows:

1. Other than in the X Protocol specification, in those cases where they restrict identifiers to using the Latin-1 alphabet, they should be amended so that the restriction is to characters of the Host Portable Character Set.
2. The X Toolkit Intrinsics specification (Section 3.6.1) should state clearly that the Host Portable Character Set shall be used for string identifiers.
3. The definition of the **String** data type should be referred to in the index of the specification in which it appears, so that it can be found easily. If it is actually missing then it should be added to the appropriate XCDE specification. It should state clearly that the data type is used for null-terminated strings of 8-bit bytes.

---

5. This protocol is not defined in the XCDE specifications. It is a Sun Microsystems specification and is available from CDE vendors. It satisfies the X/Open criteria for a proprietary specification that is relied on by an X/Open specification; the technology can be licenced on reasonable terms and is available from multiple sources.

5.13 Configuration File Syntax

The syntax of configuration files is generally English-like (see Section 3.3.3.4 on page 21, Section 4.3.9.3 on page 38, Section 4.4.3.3 on page 47 and Section 4.4.3.6 on page 48). Accelerators are described by **String** resources in a format similar to that used by the translation manager (see Section 4.1.3.3 on page 28).

The present situation is satisfactory, and no change is proposed.

5.14 Font Representations

As for configuration files (see Section 5.13), BDF font representations have an English-like syntax (see **Font Representations** on page 25). They are designed to be human-readable as well as machine-readable. From the human-readability point of view, the representations should ideally be locale-dependent. However, this is outweighed by the importance of a common machine-readable format for system interoperability.

No action is therefore proposed in respect of this issue.

5.15 Uil String Formats

It is not clear whether the strings processed and generated by *Uil*( ) are locale-dependent (see Section 4.1.3.4 on page 28).

The strings should be locale-dependent, and the XCDE specifications should be amended to say so.

5.16 Command Strings

There are some instances where command strings used to invoke applications are assumed to use the characters of the X Portable Character Set (see Section 3.2.3.8 on page 19 Section 4.1.3.2 on page 28).

This is acceptable, and no change is proposed.

5.17 Text Drawing

Programmers writing internationalised applications should take care to use the appropriate text drawing primitives (see Section 3.1.2.6 on page 16).

Use of the single-byte-character primitives should be deprecated in the XCDE specifications.

5.18 String Manipulation

String comparison and manipulation operations are likely to produce counter-intuitive results in locales where glyphs can have alternative representations (for example, <e-acute> and <e> plus <acute accent>). See Section 4.1.3.6 on page 28, **String Manipulation** on page 33, **String Manipulation** on page 39 and Section 4.4.3.4 on page 47.

The specifications should draw their readers' attention to this.

5.19 Cut Buffers

Cut buffers can only contain Latin-1 alphabet text. Internationalised applications can therefore not use the ''cut buffer'' facility. (See Section 3.2.3.10 on page 19 and **Cut Buffers** on page 22.)

Since the cut buffer mechanism is obsolescent, no action is proposed in respect of this.

5.20 Drag-and-drop Text

Textual data transferred using drag-and-drop services can be either Latin-1 alphabet text or compound text (see **Drag-and-drop Text** on page 32). The availability of a special Latin-1 text facility is an extra convenience to applications written specifically for Latin-1 locales; the availability of COMPOUND_TEXT data allows internationalised applications to be written.

No action is proposed in respect of this issue.

5.21 Character Set Registry

The X Character Set Registry appears to duplicate some of the functions of the X/Open Locale Registry (see **Character Set Registry** on page 23 and **Character Set Registry** on page 25).

No formal links between the registries are necessary. X/Open should, however, be aware of the X Character Set Registry, and of other registries such as those maintained by ECMA and OSF.

5.22 Scale Widget Number Formats

It is not clear whether the format of the numbers displayed by *XmScale* widgets is fully internationalised (see Section 4.1.3.5 on page 28).

The specifications of these widgets should be amended to require the use of the numeric and decimal separators of the current locale, and of national numbers where appropriate.

5.23 Text Widget Values

The degree to which *XmText* and *XmTextField* widgets can be used in internationalised applications is unclear (see Section 4.1.3.8 on page 28).

The descriptions of these widgets should clarify the extent to which their behaviour is locale-dependent and should explain how the **XmNValue** resource is encoded. Relevant clarifications are contained in Chapter 11 of the Athena Motif 1.1 Specification.

5.24 Locale Conflicts

There are a number of instances where information that has been created in one locale can be processed in another. They can be considered in two groups:

1. See **Locale Conflicts** on page 31, **Command and Filename Encodings** on page 44 and **Locale Conflicts** on page 37.

   These conflicts exist but there is nothing that can be done to resolve them. If UTF-8 was used for all information passed between locales then there would be no problem, but UTF-8 is not supported by all platforms.

   For the next revision of the XCDE specifications, the recommendations of the X/Open **DIS, Version 2** Snapshot should be considered, particularly the recommendations relating to the use of UTF-8.

2. See Section 4.4.3.1 on page 46.

   The user will see ''garbage'' if he tries to access a calendar using the wrong locale. Ideally:

   — Calendars should have locale attributes rather than character set attributes.

   — Individual entries should have locale attributes.

   — When entries are displayed, it may sometimes be desirable to take account of timezones when displaying dates and times.

   These issues should be considered as possible future work items.

5.25 Boolean Strings

The *DtDtsIsTrue*( ) function tests a string for a Boolean value. The strings that it identifies as meaning ''true'' are all English words (or the number ''1''). See **Boolean Strings** on page 33. This affects the application programmer rather than the user.

No action is proposed in respect of this issue.

5.26 Locale Dependence of Descriptions and Labels

There is no requirement that the DESCRIPTION field of the actions database should be locale-dependent, but the LABEL field is locale-dependent (see **Locale Dependence of Descriptions and Labels** on page 34).

This situation was intended by the XCDE developers. The DESCRIPTION field is never seen by the user. Currently, it is probably never used.

No change is proposed in respect of this issue.

5.27 Description of dtksh

The description of the **dtksh** utility leaves a number of internationalisation aspects unclear (see Section 4.3.12.2 on page 40).

The XCDE specifications state that **dtksh** behaves in the same way as XPG4 **sh**. The internationalised behaviour of this is not very clearly stated in the XPG4 specifications, but it is in fact internationalised in the same way as the POSIX.2 standard (the XPG4 Commands and Utilities Component Definition refers to this as an overriding standard).

The description of **dtksh** should be enhanced to point out this implication of the reference to XPG4 **sh**.

5.28 Terminal Emulation Issues

It is not clear how far languages that do not use the ASCII character set, and complex text languages in particular, can be supported with the defined terminal emulation (see Section 4.3.13.3 on page 41).

Ideally, it might be desirable to define emulations of other terminals that are designed to cater for non-Latin character sets, but this would not be practical. No action is proposed.

5.29 Source Code of dtcodegen

It is not clear whether the source codeset of **dtcodegen** programs is locale-dependent (see **Source Code of dtcodegen** on page 42). This affects the application developer, but not the user.

The source codeset of **dtcodegen** programs is the same as that of the C compiler in use. This should be obvious and there is no need to mention it in the XCDE specifications.

5.30 Object Palette Limitations

The object palette includes standard Text Pane and Text Field objects, but does not include standard date, time, number or money objects (see **Object Palette Limitations** on page 42).

Consideration should be given to defining standard, locale-dependent, date, time, number and money objects as possible future enhancements, not only for the object palette, but for Motif also.

5.31 Building Help Text

It is not clear whether message catalogue source file generation applies to help text (see **Building Help Text** on page 42).

The X/Open **XCDE: Services and Applications** Specification should state clearly that message catalogue source file generation does not, and should not, apply to help text.

5.32 Help Information and Actions

There is no requirement for the help information associated with action icons to be locale-dependent (see Section 4.3.17.3 on page 43).

A requirement for the help information to be locale-dependent should be added to the X/Open **XCDE: Services and Applications** Specification.

5.33 Calculator Internationalisation

It is not entirely clear what it means for an implementation of calculator services to be internationalised (see Section 4.3.19.3 on page 44).

The X/Open **XCDE: Services and Applications** Specification should be enhanced to make it clear. The enhancement should take the form of an additional capability specified for the calculator.

5.34 Mail Message Header Fields

Users are forced to use ASCII for the header fields of their mail messages (see **Mail Message Header Fields** on page 37). This is a consequence of the adoption of the RFC 822 Internet Specification format.

No action is proposed in respect of this issue.

5.35 Mail Aliases

The X/Open **XCDE: Services and Applications** Specification does not require alias lists to be localised (see **Mail Aliases** on page 37).

This is a desirable facility for future specifications.

5.36 Date, Time and Number Formats

The X/Open **XCDE: Services and Applications** Specification does not require a properly localised format for dates, times and numbers specified to the calendaring and appointment services (see Section 4.4.3.2 on page 47).

It should do so.

5.37 Encoding of csa_logon( ) Arguments

It is not clear what character set and encoding is used for the arguments of *csa_logon*( ) (see Section 4.4.3.5 on page 47).

The X/Open **XCS** Specification should state that the character sets of the arguments are the implementation-specific default character set for the implementation.

Index