X/Open Technical Study

Internationalisation of X/Open Specifications:
Special Draft Sept 1995 for XTP TWG review

X/Open Company Ltd.

© December 1994, X/Open Company Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

X/Open Technical Study

Internationalisation of X/Open Specifications: Special Draft Sept 1995 for XTP TWG review

ISBN: 1-85912-080-6

X/Open Document Number: E408

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 either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk with the following in the Subject line:

```
request corrigenda; topic index
```

This will return the index of publications for which Corrigenda exist.

This Document

This document is a Technical Study (see above). It identifies the implications of internationalisation requirements on the following types of X/Open specification:

- Interworking Specifications
- Data Management Specifications
- Distributed Transaction Processing (DTP) Specifications
- Systems Management Specifications.

The structure of this technical study is as follows:

- Part 1 — Internationalization Overview
  - Chapter 1 is an introduction to this technical study.
  - Chapter 2 discusses the subject of internationalisation in general terms and describes the provisions that have been made for it in international standards and in the X/Open CAE.
- Part 2 — X/Open Interworking Specifications
  - Chapter 3 introduces the X/Open interworking specifications that are considered by this technical study.
  - Chapter 4 describes the criteria that have been followed in identifying internationalisation issues in the X/Open interworking specifications.
  - Chapter 5 analyses the implications of internationalisation on the X/Open interworking specifications.
  - Chapter 6 presents conclusions and recommendations.
  - Chapter 7 contains a set of proposed *internationalisation* Change Requests (CRs) for the X/Open interworking specification.
- Part 3 — X/Open Data Management Specifications
  - Chapter 8 introduces the X/Open data management specifications that are considered by this technical study.

- — Chapter 9 discusses general internationalisation issues that are associated with SQL.
- — Chapter 10 analyses the implications of internationalisation on the X/Open data management specifications.
- — Chapter 11 presents conclusions and recommendations.

- Part 4 — X/Open DTP Specifications
  - — Chapter 12 introduces the X/Open Distributed Transaction Processing (DTP) specifications that are considered by this technical study.
  - — Chapter 13 analyses the implications of internationalisation on the X/Open DTP specifications.
  - — Chapter 14 presents conclusions and recommendations.
  - — Chapter 15 contains a set of proposed *internationalisation* Change Requests (CRs) for the X/Open DTP specification.
- Part 5 — X/Open Systems Management Specifications
  - — Chapter 16 introduces the X/Open systems management specifications that are considered by this technical study.
  - — Chapter 17 analyses the implications of internationalisation on the X/Open systems management specifications.
  - — Chapter 18 presents conclusions and recommendations.
  - — Chapter 19 contains a set of proposed *internationalisation* Change Requests (CRs) for the X/Open systems management specification.
- Part 6 provides a glossary and an index.

Intended Audience

This technical study is aimed in general at all users and suppliers who wish to understand the issues surrounding internationalisation and how these can be solved. In particular, it is aimed at implementors and application developers who use X/Open's interworking, data management, DTP and systems management specifications.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — variable names, for example, substitutable argument prototypes and environment variables
  - — C-language functions; these are shown as follows: *name*( )
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a C-language header file.
- Names surrounded by braces, for example, {ARG_MAX}, represent symbolic limits or configuration values, which may be declared in appropriate C-language header files by

means of the C **#define** construct.

- The notation [ABCD] is used to identify a coded return value in C.
- Syntax and code examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width`* font.

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.

Referenced Documents

The following standards are referenced in this technical study:

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

ASN.1
: ISO 8824: 1990 Information Technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).

ASN.1 DIS
: ISO DIS 8824: 1992-1993 Information Technology — Open Systems Interconnection — Abstract Syntax Notation One (ASN.1) — Specification of Basic Notation.

CMISP
: ISO / IEC 9596-1: 1991, Common Management Information Service Protocol.

FIPS 127-2
: US National Institute of Standards and Technology (NIST), Federal Information Processing Standard, FIPS 127-2, Database Language SQL.

GDMO
: ISO / IEC 10165-4:1992, Information Technology — Open Systems Interconnection — Structure of Management Information — Part 4: Guidelines for the Definition of Managed Objects.

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

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 / IEC 8571
: ISO / IEC 8571, Information Processing Systems — Open Systems Interconnection — File Transfer, Access and Management.

    Part 1: General Introduction (1988)
    Part 2: Virtual Filestore Definition (1988)
    Part 3: File Service Definition (1988)
    Part 4: File Protocol Specification (1988).

ISO 8859
: ISO 8859: 1987 Information Processing — 8-bit Single-byte Coded Graphic Character

Part 1, Latin Alphabet No. 1 (1987)
Part 2, Latin Alphabet No. 2 (1987)
Part 3, Latin Alphabet No. 3 (1988)
Part 4, Latin Alphabet No. 4 (1988)
Part 5, Latin / Cyrillic Alphabet (1988)
Part 6, Latin / Arabic Alphabet (1987)
Part 7, Latin / Greek Alphabet (1987)
Part 8, Latin / Hebrew Alphabet (1988)
Part 9, Latin Alphabet No. 5 (1989).

ISO / IEC 9594

ISO / IEC 9594: 1990, Information Technology — Open Systems Interconnection — The Directory, Parts 1 to 8:

Part 1: Overview of Concepts, Models and Services (1990)
Part 2: Models (1990)
Part 3: Abstract Service Definition (1990)
Part 4: Procedures for Distributed Operation (1990)
Part 5: Protocol Specifications (1990)
Part 6: Selected Attribute Types (1990)
Part 7: Selected Object Classes (1990)
Part 8: Authentication Framework (1990)

ISO / IEC 10021

ISO / IEC 10021, Information Processing Systems — Text Communication — Message Oriented Text Interchange System:

Part 1: System and Service Overview (1990)
Part 2: Overall Architecture (1990)
Part 3: Abstract Service Definition Conventions (1990)
Part 4: Message Transfer System: Abstract Service Definition and Procedures (1990)
Part 5: Message Store: Abstract Service Definition (1990)
Part 6: Protocol Specifications (1990)
Part 7: Interpersonal Messaging System (1990).

ISO / IEC ISP 10607-2

ISO / IEC ISP 10607-2: 1990, Information Technology — International Standardized Profiles AFTnn — File Transfer, Access and Management — Part 2: Definition of Document Types, Constraint Sets and Syntaxes.

ISO / IEC 10646

ISO / IEC 10646-1: 1993, Information Technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

ISO C

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

ISO MSE

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

ISO RDA

RDA Generic

ISO / IEC 9579-1: 1993, Information Technology — Open Systems Interconnection — Remote Database Access — Part 1: Generic Model, Service, and Protocol.

RDA SQL Specialization
: ISO/IEC 9579-2: 1993, Information Technology — Open Systems Interconnection — Remote Database Access — Part 2: SQL Specialization.

ISO SQL
: ISO/IEC 9075: 1992, Information Technology — Database Language SQL (technically identical to ANSI standard X3.135-1992).

POSIX.1
: ISO/IEC 9945-1: 1990 (also IEEE Std. 1003.1:1990), Portable Operating System (POSIX) — Part 1: System Application Program Interface.

POSIX.2
: IEEE Std. 1003.2: 1992, Portable Operating System (POSIX) — Part 2: Shell and Utilities.

SNMP
: Internet RFC 1157, The Simple Network Management Protocol.

SQL3
: ISO-ANSI Working Draft — Database Language SQL, May 1993.

T.61
: CCITT Recommendation T.61: 1984, Character Repertoire and Coded Character Sets for the International Teletex Service.

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

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

X.400
: CCITT Recommendations X.400-X.420: 1988, Data Communications Networks — Message Handling Systems. These recommendations are technically aligned with ISO 10021.

X.500
: CCITT Recommendations X.500-X.521: 1988, Data Communications Networks — Directory. These recommendations are technically aligned with ISO 9594.

The following X/Open documents are referenced in this technical study:

BSFT
: X/Open CAE Specification, December 1991, Byte Stream File Transfer (BSFT) (ISBN: 1-872630-27-8, C194).

CLI
: X/Open Snapshot, October 1992, Data Management: SQL Call Level Interface (CLI) (ISBN 1-872630-63-4, S203).

CORBA
: X/Open CAE Specification, August 1994, Common Object Request Broker: Architecture & Specification, (ISBN: 1-85912-044-X, C432).

CPI-C, Version 2
: X/Open CAE Specification, October 1995, Distributed Transaction Processing: The CPI-C Specification, Version 2 (ISBN: 1-85912-135-7, C419).

DISS, Issue 1
X/Open Snapshot, November 1992, Distributed Internationalisation Services (ISBN: 1-872630-75-8 S213).

DTP
X/Open Guide, November 1993, Distributed Transaction Processing: Reference Model, Version 2 (ISBN: 1-85912-019-9, G307).

FSS-UTF
X/Open Preliminary Specification, May 1993, File System Safe UCS Transformation Format (FSS-UTF) (ISBN: 1-872630-96-0, P316).

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

Interworking Internationalization
X/Open Snapshot, May 1993, Internationalization of Interworking Specifications (ISBN 1-872630-87-1, S302).

IPC SMB
X/Open Developers' Specification — IPC Mechanisms for SMB, XO/CAE/91/500, The X/Open Co. Ltd, 1991.

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).

(PC)NFS
X/Open Developers' Specification, August 1990, Protocols for X/Open PC Interworking: (PC)NFS (ISBN: 1-872630-00-6, D030 or XO/DEV/90/030).

RDA
X/Open CAE Specification, August 1993, Data Management: SQL Remote Database Access (ISBN: 1-872630-98-7, C307).

SMB
X/Open CAE Specification, October 1992, Protocols for X/Open PC Interworking: SMB, Version 2 (ISBN: 1-872630-45-6, C209).

SQL
X/Open CAE Specification, August 1992, Structured Query Language (SQL) (ISBN: 1-872630-58-8, C201).

TX
X/Open Preliminary Specification, October 1992, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN: 1-872630-65-0, P209).

TxRPC
X/Open Preliminary Specification, July 1993, Distributed Transaction Processing: The TxRPC Specification (ISBN: 1-85912-000-8, P305).

UCS
X/Open Technical Study, February 1994, Universal Multiple-Octet Coded Character Set

Coexistence and Migration (ISBN: 1-85912-031-8, E401).

UMA

Forthcoming X/Open Guide, expected to be published in 1995, Guide to Universal Measurement Architecture (UMA), (ISBN 1-85912-073-3, G414).

The version of UMA used during this technical study was X/Open's pre-publication draft dated October 6, 1994.

UMA DCI

Forthcoming X/Open Preliminary Specification, expected to be published in 1995, Systems Management: UMA Data Capture Interface, (ISBN 1-85912-068-7, P434).

The version of UMA DCI used during this technical study was X/Open's pre-publication draft dated November 1, 1994.

UMA DPD

Forthcoming X/Open Preliminary Specification, expected to be published in 1995, Systems Management: UMA Data Pool Definitions, (ISBN 1-85912-069-5, P435).

The version of UMA DPD used during this technical study was X/Open's pre-publication draft dated November 7, 1994.

UMA MLI

Forthcoming X/Open Preliminary Specification, expected to be published in 1995, Systems Management: UMA Measurement Layer Interface, (ISBN 1-85912-072-5, P426).

The version of UMA MLI used during this technical study was X/Open's pre-publication draft dated November 7, 1994.

XA

X/Open CAE Specification, December 1991, Distributed Transaction Processing: The XA Specification (ISBN: 1-872630-24-3, C193 or XO/CAE/91/300).

XA+

X/Open Snapshot, July 1994, Distributed Transaction Processing: The XA+ Specification, Version 2 (ISBN: 1-85912-046-6, S423).

XAP PS

X/Open Preliminary Specification, June 1992, ACSE/Presentation Services API (XAP) (ISBN: 1-872630-53-7, P203).

XAP

X/Open CAE Specification, September 1993, ACSE/Presentation Services API (XAP) (ISBN: 1-872630-91-X, C303).

XAP-ROSE

X/Open Preliminary Specification, December 1993, Remote Operations Service Element (XAP-ROSE) API, (ISBN 1-872630-86-3, P302).

XAP-TP

X/Open Preliminary Specification, February 1994, ACSE/Presentation: Transaction Processing API (XAP-TP) (ISBN: 1-872630-85-5, P216).

XATMI

X/Open Preliminary Specification, July 1993, Distributed Transaction Processing: The XATMI Specification (ISBN: 1-872630-99-5, P306).

XBSA

Forthcoming X/Open Preliminary Specification, expected to be published in 1995, Systems

Management: Backup Services API (XBSA), (ISBN 1-85912-056-3, P424).

The version of XBSA used during this technical study was X/Open's pre-publication draft dated October 7, 1994.

XDS
X/Open CAE Specification, November 1991, API to Directory Services (XDS) (ISBN: 1-872630-18-9, C190 or XO/CAE/91/090).

XDS, Issue 2
X/Open CAE Specification, February 1994, API to Directory Services (XDS), Issue 2 (ISBN: 1-85912-007-5, C317).

XFTAM PS
X/Open Preliminary Specification, September 1992, FTAM High-level API (XFTAM) (ISBN: 1-872630-60-X, P206).

XFTAM
X/Open CAE Specification, January 1994, FTAM High-level API (XFTAM) (ISBN: 1-85912-010-5, C304).

XGDMO
X/Open Preliminary Specification, March 1994, Systems Management: GDMO to XOM Translation Algorithm (ISBN: 1-85912-023-7, P319).

XMP
X/Open CAE Specification, March 1994, Systems Management: Management Protocol API (ISBN 1-85912-027-X, C306).

XMPP
X/Open CAE Specification, October 1993, Systems Management: Management Protocol Profiles (ISBN 1-85912-018-0, C206).

XMPTN Access Node
X/Open Preliminary Specification, September 1994, Multiprotocol Transport Networking (MPTN): Access Node, (ISBN 1-85912-040-7, P408).

XMPTN Address Mapper
X/Open Preliminary Specification, September 1994, Multiprotocol Transport Networking (MPTN): Address Mapper (ISBN 1-85912-039-3, P407).

XMS
X/Open CAE Specification, June 1993, Message Store API (XMS) (ISBN: 1-872630-83-9, C305).

XSMS
Forthcoming X/Open Preliminary Specification, expected to be published in 1995, Systems Management: Management Services in an OMG Environment, (ISBN 1-85912-047-4, P421).

The version of XSMS used during this technical study was X/Open's pre-publication draft dated November 1, 1994.

XNFS
X/Open CAE Specification, October 1992, Protocols for X/Open Interworking: XNFS, Issue 4 (ISBN: 1-872630-66-9, C218).

XOM
X/Open CAE Specification, November 1991, OSI-Abstract-Data Manipulation API (XOM) (ISBN: 1-872630-17-0, C180 or XO/CAE/91/080).

XOM, Issue 2

X/Open CAE Specification, February 1994, OSI-Abstract-Data Manipulation API (XOM), Issue 2 (ISBN: 1-85912-008-3, C315).

XPG1

X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).

XPG2

X/Open Portability Guide, five volumes, January 1987 (ISBN: 0-444-70179-6).

XPG3

X/Open Specification, 1988, 1989, February 1992 (ISBN: 1-872630-43-X, T921); this specification was formerly X/Open Portability Guide, seven volumes, January 1989 (ISBN: 0-13-685819-8, XO/XPG/89/000).

XPG4

XBD, Issue 4

X/Open CAE Specification, July 1992, System Interface Definitions, Issue 4 (ISBN: 1-872630-46-4, C204).

XCU, Issue 4

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

XSH, Issue 4

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

XRM

X/Open Guide, August 1993, Systems Management: Reference Model (ISBN: 1-85912-05-9, G207).

XTI

X/Open CAE Specification, January 1992, X/Open Transport Interface (XTI) (ISBN: 1-872630-29-4, C196 or XO/CAE/91/600).

X.400 API

X/Open CAE Specification, December 1991, API to Electronic Mail (X.400) (ISBN: 1-872630-19-7, C191 or XO/CAE/91/100).

X.400 API, Issue 2

X/Open CAE Specification, February 1994, API to Electronic Mail (X.400), Issue 2, (ISBN: 1-85912-009-1, C316).

X/Open Technical Study

Part 1

Internationalisation Overview

X/Open Company Ltd.

Chapter 1

Introduction

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. They affect the networking technology on which distributed systems are based, and also the methods by which data is stored, manipulated and administered on behalf of users. If computer systems do not take the requirements of internationalisation into account, the ability of a distributed system to work in different languages and cultural environments is limited.

This technical study identifies the implications of internationalisation requirements on the following types of X/Open specification; each of which is discussed in more detail in the following parts of this document:

- Interworking Specifications — see Part 2
- Data Management Specifications — see Part 3
- Distributed Transaction Processing (DTP) Specifications — see Part 4
- Systems Management Specifications — see Part 5.

Before these four parts, Chapter 2 discusses the general subject of internationalisation, and in particular describes the provisions that are made for it in international standards and in the X/Open Common Applications Environment (CAE).

Chapter 2

Internationalisation

2.1 Overview

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 open systems environment 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 open systems environment 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. However, 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 decimal-point (.) as 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 open systems environment 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.

2.2 Character Sets and Encodings

A wide variety of character sets is used to represent the languages of the world. This document 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 (now the ITU-T) 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 the referenced ISO 646 international standard, 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 the referenced ISO 2022 international standard. 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 the referenced ISO 2375 international standard. 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 the referenced ISO 8859 international standard.

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

It should be noted that all the above standards use the same encodings as the referenced ISO 646 international standard 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 10646 (which incorporates the work of the UNICODE consortium), departs from these principles.

ISO 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 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 which is 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 which is 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 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 the referenced ISO 10646 international standard; the referenced UNICODE standard is equivalent to just one of these ways: UCS-2 Level 3.

A degree of compatibility with ISO 646 is maintained, in that the characters encoded by ISO 646 are encoded by the ISO 10646 using the ISO 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 the ISO 646, and as (hex) 00 41 by ISO 10646.

However, the ISO 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 10646, or as any of the three leading octets of an encoding of the UCS-4 form. For example, the ISO 646 encoding of the End of Text (<ETX>) character appears as an octet of the ISO 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 10646 control codes.

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

The algorithm defined by ISO 10646 (known as UTF-1) does not prevent encodings from containing the ISO 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, known as FSS-UTF or UTF-8, has therefore been defined by the X/Open-Uniforum Joint Internationalisation Group (JIG). In this UTF, an octet with bit 8 set to zero can only appear as the single-octet representation of the identical ISO 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.

A full discussion of the issues pertaining to the use of ISO 10646 in Open Systems is contained in the referenced **UCS** technical study.

2.3 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 open systems environment, 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 the referenced X/Open **XPG1** Portability Guide, 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 the referenced ISO C international standard 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 referenced ISO MSE addendum 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 ISO MSE addendum includes function *wcscat*( ), 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 the referenced ISO C international standard in conjunction with the coded character set defined in the referenced ISO 10646 international standard is problematic.

In addition to permitting flexibility of character sets and encodings, ISO C specifies a *locale* mechanism which can be used to enable applications programs to be written without making assumptions about language and cultural conventions. ISO 9899 (the ISO C Standard) 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. The ISO C standard 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.

2.4 Internationalisation Support in POSIX

The locale mechanism of ISO C is extended by the referenced POSIX.1 international standard[1]. 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, POSIX.1 still specifies the general mechanism only, and contains no standardised descriptions of specific locales (other than the default locale).

Also, POSIX.1 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 0-9 and the period, underscore and hyphen characters (as found in ISO 646).

The interface specified in the referenced POSIX.2 IEEE standard[2] 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 referenced POSIX.1 international standard. It includes additional punctuation characters such as *{* and *}*.

Several of the utilities defined in the referenced POSIX.2 IEEE 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[3].

---

1. The ISO 9945-1:1990 POSIX.1 standard is identical to IEEE Standard 1003.1-1990. It specifies a programming interface to operating system services.
2. IEEE 1003.2-1992 specifies a user interface to operating system services (commands and utilities).
3. 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.

2.5 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 applications 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 referenced ANSI C standard. They were expanded and refined in Issue 4 (the referenced **XPG4-XSH** X/Open CAE specification) including full conformance with ISO C. (ISO C is based on, and technically equivalent to, ANSI C.)

A more complete description of the development of internationalisation facilities can be found in the referenced X/Open **Internationalisation Guide**. The differences between Issue 3 and Issue 4 of the XSI are summarised in the referenced **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 2.5.1.

Recent further work within the X/Open-Uniforum Joint Internationalisation Group (JIG) has been concerned with internationalisation within a distributed systems environment. This concludes that the internationalisation facilities specified in the referenced **XPG4-XSH** X/Open CAE 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 2.5.2.

2.5.1 XPG4 Facilities

Firstly, the referenced **XPG4-XSH** X/Open CAE specification includes the **wchar_t** type of ISO C and the locale mechanism of the referenced POSIX.1 international 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 referenced ISO MSE addendum to ISO C.

While the referenced **XPG4-XSH** X/Open CAE 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.

2.5.2 Distributed Internationalisation Requirements

The X/Open-Uniforum Joint Internationalisation Group (JIG) has produced the referenced **DISS Issue 1** X/Open snapshot. This document discusses the issues arising from the need for internationalised applications programs executing in a distributed environment. In particular, 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 instance). 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 **DISS** contains a detailed proposal for a standardised way of describing locales, and proposes that a registry of standard locales should be established. Methods of conveying locale information between distributed applications are still being studied.

The DISS also proposes a set of functions for processing *self-announcing data*. Such data may include indications of the locale or locales in which it should be processed. (The term *tagged data* has also been used). The use of self announcing data would enable the applications to use the appropriate locale or locales. This would support *multi-locale* applications that handle information from several different locales at the same time.

2.6 Current Work

Work is continuing on the following topics.

2.6.1 Revision of the DISS

The **DISS** is being revised in the light of developments. The latest draft is significantly changed from the published snapshot. The set of functions defined has been changed substantially and provides consistent and comprehensive multi-locale support.

2.6.2 Definition and Registration of Locales

A registry of standard locales has been established by X/Open. The operation of the registry is described in the X/Open **Locale Registry** Procedures guide. The locales in the registry can be obtained from X/Open. At the time of writing, the registry contains some 20 locales, including Danish, Dutch, English (American and British), Faroese, German (Austrian, German and Swiss), Greenlandic, Hungarian, Icelandic, Italian, Japanese, Latvian, Lithuanian, Polish, Portuguese and Romanian locales.

2.6.3 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.

The current state of work on complex text languages is embodied in the referenced **Layout Services** snapshot. This explains the difficulties associated with processing text written in these languages, and describes some facilities that could be added to the X/Open CAE to help overcome them:

- an opaque data structure (a layout object) that can be associated with a locale and that describes the characteristics of a piece of text written in a complex text language
- functions that:
  - manipulate layout objects

    and
  - transform text in accordance with the characteristics given in layout objects

- a new locale category (LO_LTYPE) which could be implemented as:
  — an extended version of the LC_CTYPE category
  or
  — as part of the layout object data structure.

2.6.4 Use of UNICODE/ISO 10646

ISO 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 a standard locale, or perhaps several standard locales, that use the ISO 10646 encoding (or perhaps a related UTF encoding) be defined?
- Should all implementations support all characters defined in the referenced ISO 10646 international standard (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?
- How should APIs (and particularly C language APIs) provide for character strings that may be encoded in accordance with ISO 10646?
- Should ISO 10646, or perhaps a UTF, be specified as the standard encoding for use in certain situations in Open Systems?
- Should UCS-dependent APIs (that is, APIs that assume that character data is encoded in accordance with ISO 10646 UCS-2) be defined?

2.6.5 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.

2.6.6 Distributed Internationalisation Framework

A framework document that sets the context for work on internationalisation in distributed systems is in preparation. It will provide an overview and analysis of the problem areas, but will not contain detailed interface specifications, which will be in the DISS.

X/Open Technical Study

Part 2

X/Open Interworking Specifications

X/Open Company Ltd.

Chapter 3

Introduction

The chapters in this part of this technical study consider the impact of internationalisation on the X/Open interworking specifications. These chapters supersede the previously published X/Open **Interworking Internationalisation** snapshot.

The X/Open interworking specifications examined in this Technical Study are those that at the time of writing are already, or are expected shortly to become, CAE specifications. They consist of the following documents (full details are given in **Referenced Documents** on page xv).

- the **XTI** specification
- the **XMPTN** specifications (Access Node and Address Mapper)
- the **XAP**, **XAP-TP**, and **XAP-ROSE** specifications
- the **XOM** specification
- the **XFTAM** specification
- the **BSFT** specification
- the **X.400 API** specification
- the **XMS** specification
- the **XDS** specification
- the **XNFS** specification
- the **(PC)NFS** specification
- the **SMB** Protocols specification
- the **IPC** Mechanisms for SMB.

Structure of This Part

Chapter 4 describes the criteria that have been followed in identifying internationalisation issues in the X/Open interworking specifications.

Chapter 5 examines the implications of internationalisation on the interworking specifications listed above.

Chapter 6 presents conclusions and recommendations.

After this, Chapter 7 contains a set of *internationalisation* Change Requests (CRs) for the interworking specifications listed above. These are edited versions of standard X/Open Change Requests (CRs), in which the identity of the originator is omitted and the CRs are re-numbered into a sequential scheme, for the purposes of this document.

Chapter 4

Potential Issues for Interworking

For the purposes of this technical study, occurrences in an X/Open interworking specification of either of the following requirements has been taken as a sign that there may be an internationalisation issue:

- A requirement for a particular character set or encoding (for example, ASCII) for textual information passed across an API or over a communications interface.

  Note: A requirement for representation of non-textual information in a specific way has not been considered to give rise to internationalisation issues. For example, a date in ddmmyy form can be interpreted by a program operating in any language environment. It is only when that form is used at the user interface that there are problems. Similarly, a requirement for a constant text string expressed in a particular language (such as ''LANMAN'') does not pose internationalisation problems since it is an arbitrary value that can be processed by programs as a constant.

- A requirement for any form of user interface.

  Although it is possible to specify a user interface in an internationalised way, there are so many problems in this area that it is right to analyse any user interface requirement.

In considering the internationalisation issues that may arise from occurrences of the above requirements the following four criteria have been used.

Specification Portability

Can the specification be meaningfully implemented in any language and cultural environment?

For example, a requirement for a user interface which requires particular English language commands and responses does not meet this criterion. Nor does a requirement for an API in which the application must pass text encoded as 7-bit ASCII strings.

Applications Portability

Assuming portable specifications, can applications be written in such a way that they can operate under different language and cultural environments?

For example, a specification that requires text to be passed to the application *in the local language* can be implemented in any language environment but will not support internationally portable applications.

Implementation Interworking

Can implementations written in different language and cultural environments interwork?

For example, a specification that requires use of different national character set encodings derived from a single international superset would not necessarily produce implementations that interwork since there could be situations where a character code is legally usable in one country but not in another.

Applications Interworking

Can an application interwork with other applications or service implementations on remote machines operating under different language and cultural environments?

For example, interworking could be a problem where one application sorts text generated by another application on another system and the service does not enable the sorting application to determine the collating sequence to be used. Note that this issue can arise where two applications interwork using a service (for example, they communicate using XTI) or where an application interworks with a service in a remote system (for example, where an application uses XFTAM).

Although use of the POSIX Portable Filename Character Set is recommended in POSIX, it does restrict users whose natural language is not English. For example, a Dane could not use his spelling of the name which in English is spelt *Aarhus*. For this reason, although they may provide applications portability, character set restrictions are identified in this technical study as affecting specification portability.

Chapter 5

Interworking Specifications

5.1 The XTI Specification

5.1.1 Overview

The X/Open Transport Interface (**XTI**) Specification defines an API to a transport service that provides process-to-process communication and can be used in connection with OSI transport protocols, with TCP or UDP from the Internet Protocol Suite, with OSI Transport over TCP as defined in RFC 1006, with a minimal 7-layer OSI stack, with X.25, with SNA or with NETBIOS.[4]

5.1.2 Internationalisation Implications

The t_error() Function

The *t_error*( ) function defined in Chapter 6 of the **XTI** specification takes a null terminated character string as input, and writes to standard output an error message consisting of the input character string followed by a colon, a space, a standard error message describing the last encountered error, and a newline character. This raises a number of issues.

The input is a null terminated string. This is in accordance with the referenced ISO C international standard, but does not allow use of the referenced ISO 10646 international standard. This affects specification portability.

There is nothing to prevent, or even to discourage, the applications programmer from hard-coding the input strings into the program source code, rather than obtaining them from a catalogue associated with the currently established locale. This relates to applications portability (applications portability is not precluded, but nothing is done to encourage it).

A character set containing the newline, colon and space characters is assumed. These (in particular, the colon character) may not be meaningful in all language and cultural environments. This affects specification portability.

The input string is written to standard output followed by the standard error message. This may not be the most appropriate order in all language and cultural environments. This affects specification portability.

The standard error message strings are specified for the English language but are implementation-defined for other languages. There is nothing, however, to say that the standard error message string that is output should depend on the currently established locale. This impacts on applications portability; it will not be possible to write an internationalised application that is portable between implementations that assume different natural languages. For example, an application running on an English implementation would display English messages, even to French users.

---

4. The appendices dealing with the minimal 7-layer OSI stack, with X.25 and with SNA are yet to be published. Drafts have been reviewed for this technical study. They contain no internationalisation implications.

A change request to make the error message depend on the current locale (see Change Request XOP-1 in Chapter 7) has been accepted, and will be implemented in the next published version of the specification. A further change request to replace type **char** by type **wchar_t** in the *errmsg* argument of *t_error*( ) was not accepted. However, X/Open is to consider specifying a parallel function that will use **wchar_t** rather than **char**.

The t_strerror() Function

In the description of *t_strerror*( ) in Chapter 6 of the **XTI** specification, it is stated that a string is returned.

The string returned is null terminated. This is in accordance with the referenced ISO C international standard, but does not allow use of UNICODE or ISO 10646. This affects specification portability.

Specific text is provided for the case where the natural language is English and it is stated that ‘‘in other languages, an equivalent text is provided.’’ This impacts on applications portability (as in the case of the string generated by *t_error*( ), described above).

As for *t_error*( ), a change request to make the error message depend on the current locale (see Change Request XOP-1 in Chapter 7) has been accepted, and will be implemented in the next published version of the specification, but a further change request to replace type **char** by type **wchar_t** in the **errmsg** argument of *t_error*( ) was not accepted.

NetBIOS

In section D.5 of the **XTI** specification, the description of NetBIOS names prohibits the first octet from being null or hexadecimal FF, prohibits the last octet from being in the range (hexadecimal) 00-1F, and reserves special meaning to the ASCII code for the asterisk character. This restricts the codesets that may be used, and hence affects specification portability.

This problem derives from the NetBIOS specification, rather than the XTI specification. It is not appropriate to address it by changing XTI.

5.2 The XMPTN Specification

5.2.1 Overview

The X/Open Multi-Protocol Transport Networking (XMPTN) architecture supports mixed protocol networking. It enables an application that was designed to run over a single, specific protocol (such as SNA, NetBIOS, OSI Transport or TCP/IP) to run over additional networks using different protocols.

The XMPTN architecture includes:

- *Access Nodes*
- *Address Mappers*.

In an XMPTN *access node*, an application program that uses communications transport services (a *transport user*) interfaces, through XMPTN components, to a *transport provider* that provides those services. The transport provider may implement a communications protocol other than the one whose use is assumed by the transport user. The XMPTN components translate the transport service requests made by the user into the transport service primitives provided by the transport provider. The XMPTN components also provide compensation for services assumed by the transport user but not implemented by the transport provider, by implementing those services using the services that the transport provider does provide. The components of an XMPTN access node, and the way they operate with transport users that use TCP/IP, UDP/IP, SNA, NetBEUI and NetBIOS, are described in the referenced **XMPTN Access Node** X/Open preliminary specification.

An XMPTN *address mapper* holds details of the relationships between transport addresses used by transport users and transport addresses used by transport providers. An access node can communicate with an address mapper, using a transport service, to register address mappings and to determine the transport provider addresses corresponding to a transport user address. The function of an XMPTN address mapper is described in the referenced **XMPTN Address Mapper** X/Open preliminary specification.

5.2.2 Internationalisation Implications

There may be internationalisation implications in the address formats of some of the transport protocols mapped by XMPTN. However, the purpose of XMPTN is to allow existing protocols to continue to be used in a multi-protocol environment, rather than to provide a new transport mechanism. In general, these implications should therefore be recognised as limitations deriving from the use of the existing protocols, rather than as defects in XMPTN that should be corrected by changing XMPTN.

A slightly different case is that of TCP transport providers, for which transport user addresses may be resolved by being converted to ASCII host names and looked up using the Internet Domain Name Service. It is not explained how names expressed in codesets that do not readily convert to ASCII should be handled. It would be possible to derive coding schemes, such as that used for the algorithmic mapping used to convert IP addresses to SNA LU names, in which any address encoding can be converted to ASCII. Unless such schemes are used, there will be specification portability issues arising from the use of XMPTN with TCP transport providers.

5.3 The XAP, XAP-TP and XAP-ROSE Specifications

5.3.1 Overview

The X/Open ACSE/Presentation Services API defines an Application Program Interface to the Association Control Service Element (ACSE) of the OSI Applications Layer and the OSI Presentation Layer Service, excluding the encoding of information in ASN.1 (this is left as the responsibility of the application). It is not based on the XOM API (which provides an interface to ASN.1), although it can be used in conjunction with XOM.

The XAP-ROSE API is an extension of the XAP API that provides an interface to the OSI Remote Operation Service Elements (ROSE).

The XAP-TP API is an extension of the XAP API that provides access to the services of the OSI Transaction Processing (TP) protocol.

5.3.2 Internationalisation Implications

Service Provider Identifiers

Service providers are identified by null-terminated strings. This would cause problems for an application using the ISO 10646 codeset, and hence affects specification portability.

Diagnostic Messages

The *ap_get_env*( ) function returns a diagnostic message in field *error* of structure *ap_diag_t* when passed the AP_DIAGNOSTIC attribute. This message is in the natural language of the currently defined locale. However, it is contained in a null-terminated character string. There is therefore a question of specification portability, since null-terminated character strings can not be used in conjunction with the codeset of ISO 10646. X/Open is to consider specifying a parallel function that will use **wchar_t** rather than **char**.

Error Messages

The *ap_error*( ) function returns a pointer to an error message in the language of the currently specified locale. Again, there is a specification-portability issue, because the message is contained in a null-terminated string, and X/Open is to consider specifying a parallel function that will use **wchar_t** rather than **char**.

XAP-ROSE

There are no internationalisation implications for the referenced **XAP-ROSE** X/Open preliminary specification.

XAP-TP

There are no internationalisation implications for the referenced **XAP-TP** X/Open preliminary specification.

5.4 The XOM Specification

5.4.1 Overview

The X/Open Abstract Data Manipulation (**XOM**) specification defines a general purpose OSI Abstract Data Manipulation Application Program Interface (API) for use in conjunction with other X/Open application specific APIs for Open Systems Interconnection (OSI). XOM deals with *information objects* that arise in OSI, ie. those that can be expressed in terms of ASN.1 The specification defines how objects can be created, examined, modified and deleted.

Character String Types

The definition of ASN.1 in the referenced ISO 8824 international standard (the version on which the XOM was originally based) specifies the characters that can be represented in the various sorts of string either by explicitly defining them or by referring to character set registration numbers in ISO 2375. The types of string defined in the referenced ISO 8824 international standard are as follows:

- *General String* - all internationally registered graphic and control character sets (plus SPACE and DELETE)
- *Graphic String* - all internationally registered graphic character sets (plus SPACE)
- *IA5 String* , *VisibleString (ISO646String)*, *PrintableString* - variations of the basic ASCII character set
- *TeletexString (T61String)*, *VideotexString* - certain identified internationally registered graphic and control character sets.

Since that version, the definition of ASN.1 has been revised. Three new types of character string have been added. They are:

- *Universal Strings* - strings of characters encoded according to ISO 10646 UCS-4 form
- *BMP Strings* - strings of characters encoded according to ISO 10646 UCS-2 form
- *Unrestricted Strings* - strings of characters with a syntax that has an ASN.1 object identifier and which either
  - — are type-compatible with characters of one of the old character string types or the new Universal String type,

    or
  - — have a possible transfer syntax (that is, an encoding) that has an ASN.1 object identifier.

The Unrestricted String type allows use of any character set and encoding once the necessary object identifiers have been allocated to them. In particular, it allows use of the character sets and encodings defined in ISO 2375 and ISO 10646.

The string type definitions in XOM reference the original ASN.1 character string definitions, and also the new Universal and Unrestricted string types, but not the new BMP String type.

5.4.2 Internationalisation Implications

The referenced **XOM** X/Open CAE specification does not allow for the ASN.1 BMPString type introduced in the ISO/IEC 8824:1 1994 standard. The addition to XOM of a BMP character string syntax corresponding to this ASN.1 type should be considered.

While the string types defined in XOM include types that are capable of representing any national character set, they also include types that are restricted to representing only some national character sets. The use of these string types in APIs that use XOM thus requires careful consideration to ensure that those APIs are fully ‘‘internationalised’’. The possible use in these APIs of string types corresponding to the new Universal and Unrestricted String types consideration. These aspects are discussed in Section 5.7 on page 32 and Section 5.9 on page 34 of this technical study.

5.5 The XFTAM Specification

5.5.1 Overview

The **XFTAM** specification defines an API to the OSI File Transfer, Access and Management (FTAM) service. It supports file types FTAM-1, FTAM-2 and FTAM-3. It is defined using the X/Open OSI Abstract Data Manipulation Service (XOM).

5.5.2 Internationalisation Implications

Identifier Strings

File names, creator identities etc. are represented in the API by XOM Graphic Strings. This means that any internationally registered character set can be used. It does not allow use of UNICODE/ISO 10646, however, or have the full generality of the Unrestricted Strings of the referenced DIS 8824 draft international standard. There is therefore an impact on specification portability.

The XFTAM API reflects the referenced ISO 8571 international standard and should not be changed independently of it. Work on the FTAM standards should be kept under review, and any change in them to permit use of Universal or Unrestricted Strings should be reflected in XFTAM. At the time of writing, this issue has been raised in ISO, but there is no formal defect report, and no concrete plan to amend the FTAM standards.

File Contents

The FTAM-1 and FTAM-2 document types specify text files. Conversion of format-effector characters (in particular, *end of line*) is specified to occur on input and output. All of the XOM character string types except Universal String and Unrestricted String are supported. As noted for Identifier Strings above, this is not completely general, and there is an impact on specification portability. No change should however be made to XFTAM until a corresponding change has been made to the referenced ISP 10607-2 international standardised profile, in which the document types are defined. As with the types of file names etc. (discussed above), there is no concrete work plan to address this issue.

Although the application using XFTAM will in general be aware of the codeset used in a text file (from the Content-Class OM attribute of the Content-Type OM attribute of the FTAM-Attributes objects returned by the interface functions), it will not be aware of the collating sequences, case conversion rules etc. that apply. There is thus an implication for applications interworking. To address this, it would be necessary for locale information to be associated with the file in some way, and to be passed to the remote application. Whether this is desirable and, if so, how it could be achieved, is for further study.

Diagnostic Text

The FTAM-Output-Parameters OM class, which defines a class of OM object that is returned by several of the API functions, contains OM attribute FTAM-Diagnostic-List, which is of syntax Object(FTAM-Diagnostic), and in turn contains an OM attribute, Text-Message, which is of syntax String(Graphic) and contains an optional text message in natural language.

For applications interworking, this message should be in the natural language of the current locale. However, since the text is generated in the remote system (the FTAM responder), this would imply some means of conveying locale information between initiator and responder. This is an issue for further study.

5.6 The BSFT Specification

5.6.1 Overview

The X/Open **BSFT** Specification defines a file transfer utility similar to the IPS ftp utility but using the OSI FTAM protocol. It includes a specification of a protocol profile and a user interface.

5.6.2 Internationalisation Implications

User Interface

The user interface (defined in **BSFT** Appendix A) is specified in English. This clearly impacts on Specification Portability.

The command and argument names use only the characters of the portable filename character set of the referenced ISO 9945-1 international standard and the interface should therefore be usable wherever the character set is derived from the Latin alphabet, although the command and argument names will not be meaningful in languages other than English.

The user interface should be defined in such a way that a meaningful version of it can be derived for any natural language environment. This could be done by requiring that standard Internationalisation facilities are used for display of dates and times and for collating file names and that command and argument names are held in natural language specific message catalogues that can be referenced through locale information.

File Types

BSFT supports both text files (type FTAM-1) and binary files (type FTAM-3). There are no internationalisation issues relating to the contents of binary files, but there are two implications relating to text files.

First, as for XFTAM, the contents of text files can be of various types, including Visible String, IA5 String, Graphic String or General String, but this is not sufficiently general to cater for all possible character sets and encodings, and there is thus an implication for specification portability.

Secondly, there is a possible impact on implementation interworking relating to text files. BSFT can be used to transfer text files between two locales that use the same codeset (or that use codesets that are closely related) but will not effect a meaningful transfer of textual information between locales that use radically different codesets (for example, Hebrew and Korean). Such a transfer is clearly beyond the scope of BSFT. However, its behaviour if such a transfer is attempted is currently not defined. It should ideally be defined in a fully internationalised specification. (Note that this problem does not arise in relation to the **XFTAM** specification, since the codeset used is known to an XFTAM application from the Content-Class OM attribute, as discussed under **File Contents** on page 29 of this technical study.) This issue is not specific to interworking; the same problem can arise when transferring information between files created using different locales on a single machine. It raises the question of whether locale information could or should be associated with files in some way. In an interworking context, it also raises the question of how the locale information could be conveyed between the communicating systems.

Names

Filenames, user identities, account names and passwords can be of type Graphic String, and wildcard expansion can be requested. Again, there is an impact on specification portability, in that the possible codesets used are restricted, and there is an impact on implementation interworking, in that this allows use of BSFT between two locales with similar character sets but not between locales with different character sets. Again, such use is beyond the scope of BSFT but the behaviour of BSFT when such use is attempted should be specified.

Natural Language Text in PDUs

Protocol error messages can contain details expressed in natural language and Initialise Request/Response PDUs may include implementation information expressed in natural language. This has a possible impact on implementation interworking. If these textual messages can be displayed at the user interface (**BSFT** Appendix A does not specify either that they should be or that they should not be) then similar considerations apply as for filenames, user identities etc. In addition, even where such information is transferred between two communicating systems that use similar codesets, the user may not understand it since it may not be expressed in his natural language.

5.7 The X.400 API Specification

5.7.1 Overview

The X/Open **X.400 API** Specification defines:

- an X.400 Application API that makes the functionality of a message transfer system (MTS) accessible to a message store (MS) or user agent (UA)
- an X.400 Gateway API that divides a message transfer agent (MTA) into two software components: a mail system gateway and an X.400 gateway service.

It is defined using the X/Open OSI-Abstract-Data Manipulation service (XOM).

5.7.2 Internationalisation Implications

The API is based on the definitions of the referenced X.400 CCITT Recommendations which are essentially the same as the referenced ISO 10021 international standard. They can therefore be expected to be fully *international*. In fact, the only internationalisation implications of the API specification derive directly from the CCITT X.400 Series recommendations. This is the fact that certain class attributes are defined to be Teletex Strings, Videotex Strings or Printable Strings which (as discussed in the section on XOM) restricts them to certain, internationally registered, character sets. The permitted character sets include all Western European characters and Japanese Kanji (set nr. 87 in the ISO register) but not other sets such as Hebrew and Arabic.

In some cases (for example, that of the **Teletex Document** attribute of the **Teletex Body Part** class), this restriction is inevitable and can not be considered to impact on internationalisation. It makes no sense to use a non-teletex character set in a Teletex Body Part. In other cases, notably those of many of the attributes of class *OR-Address*, the restriction is harder to understand.

The definitions meet the needs of Western Europe, North America and Japan but not of other countries. It is not clear why the X.400 recommendations do not specify Graphic Strings, General Strings or even Octet Strings, instead of Teletex Strings. This would enable characters of any internationally registered character set to be printed. It is possible that future versions could specify the Universal, BMP or Unrestricted Strings of the new DIS 8824.

There is thus an impact on specification portability deriving not from the API specification but from the underlying X.400 series recommendations of the CCITT.

Changes should not be made in advance of changes to ISO 10021 and the X.400 series recommendations. The progress of these standards should be kept under review and the **X.400 API** Specification should be modified to reflect any changes that are made to them to address this issue.

ISO SC18 WG4 has had an outstanding work item to address this issue for the last three years. However, no national body has yet made a concrete proposal. ISO SC18 WG4 would welcome input from X/Open on the appropriate timescale for addressing the issue.

5.8 The XMS Specification

5.8.1 Overview

The X/Open XMS API provides an API to message store functions similar to those described in X.413 (see the referenced X.400 CCITT Recommendations). It is complementary to the X.400 API and, like the X.400 API, it is defined using the X/Open OSI-Abstract-Data Manipulation service (XOM). Also, it uses some of the class definitions from the X.400 and XDS APIs.

5.8.2 Internationalisation Implications

As with the X.400 API, there are a number of attributes that represent text strings and that are restricted by their syntaxes to certain character sets. These include the following:

- the **Content-Identifier** OM attribute of OM class **Auto-Forward-Arguments**, which has syntax **Printable String**
- the **IA5-String** OM attribute of OM class **Password**, which has syntax **IA5 String**
- the **A-Content-Identifier** MS attribute, which has syntax **Printable String**
- the **IM-Auto-Forward-Comment** IM attribute, which has syntax **Printable String**
- the **IM-Languages** IM attribute, which has syntax **Printable String**
- the **IM-Subject** IM attribute, which has syntax **Teletex String**
- the **IM-Suppl-Receipt-Info** IM attribute, which has syntax **Printable String**
- the **Subject** OM attribute of OM class **Heading**, which has syntax **Teletex String**.

As with the X.400 API, there is an impact on specification portability which derives from the underlying X.400 series recommendations of the CCITT, rather than from the API specification.

Changes should not be made in advance of changes to ISO 10021 and the X.400 series recommendations. The progress of these standards should be kept under review and the **XMS** API specification should be modified to reflect any changes that are made to them to address this issue.

5.9 The XDS Specification

5.9.1 Overview

The X/Open API to Directory Services (XDS) defines an API to directory services that include, but are not limited to, those defined in the referenced X.500 CCITT Recommendations. The assumed model of a Directory, and many of the associated definitions, are directly derived from those CCITT recommendations (which are aligned with and technically equivalent to the referenced ISO 9594 international standard).

The API is defined using the X/Open OSI-Abstract-Data Manipulation service (XOM).

5.9.2 Internationalisation Implications

Attribute Character Sets

For certain attributes - notably **A-Country-Name** and **A-Destination-Indicator** - there appears to be an issue because they are defined as (Latin alphabet) printable strings. This is necessary, however, in order to allow addresses to be recognisable internationally. (The **A-Country-Name** attribute, for example, is used to hold the standard country codes defined in ISO 3166). No action should therefore be taken to modify the XDS in respect of these attributes.

Attribute Comparisons

In the definitions of OM classes **Filter-Item** and **Search-Criterion**, various attribute comparison methods are discussed.

- Attribute matches can be ‘‘approximate’’ using an implementation-dependent algorithm. It is probable that full *internationalisation* requires the algorithm to take account of the locale of the user and/or the subject of the directory entry - approximate matching may well be different in England and Japan. There is thus again an impact on specification portability.

  Consideration could be given to adding a *locale* OM attribute to OM class **filter** (to allow the user’s locale to be specified) and to providing a mechanism for the algorithm to take account of any *locale* (directory) attribute in the directory entry. However, it should be noted that the definition of **filter** is based closely on definitions in CCITT Recommendation X.511. It should not be changed by X/Open until corresponding changes have been made to the Directory Standards by ISO and the CCITT.

- Greater-or-equal/less-or-equal comparisons are made using ‘‘the appropriate ordering algorithm’’. For strings, it is probable that this algorithm should use collating sequence tables for the locale of the user and/or the entry and should cater for ‘‘right-to-left’’ (eg. Arabic) as well as ‘‘left-to-right’’ text strings. Again, these rules are defined in the CCITT X.500 series recommendations and X/Open should not change the XDS until corresponding changes have been made to the Directory Standards by ISO and the CCITT.

The 1993 version of the Directory Standards includes provision for strings of ASN.1 type Universal String (but not for strings of type BMP String or Unrestricted String). Since Universal String maps to the 4-octet ISO 10646 representation rather than to the 2-octet representation (which is probably the more commonly used), there is a proposal to add support for BMP String, which does map directly to the 2-octet representation. There is currently no intention of adding support for Unrestricted character strings.

There is a further change proposed to the Directory Standards that could enable all the outstanding internationalisation issues to be resolved. This is that all directory attributes should be allowed to have a number of properties. The properties that have been considered include *language* or *locale* (*language*, rather than *locale*, seems to be favoured currently). (Other properties, such as the *time* when the validity of the attribute expires, are being considered also.)

The progress of these proposals should be monitored, and the XDS should be modified to reflect any resulting changes in the standards. It would indeed be possible to modify XDS now to take account of the new Universal String type. However, the decision on what changes to make, and when to make them, should take into account:

- the possibility that the Directory standards will be modified to cater for the BMP String type
- the other possible changes to the Directory standards discussed above
- the fact that further changes (not related to internationalisation) have been made to the Directory standards, and that it may be desirable to modify the XDS to reflect them.

5.10 The XNFS Specification

5.10.1 Overview

With the **XNFS** specification, X/Open provides a temporary but complete solution to the problem of transparent file access between X/Open-compliant systems. XNFS is described as temporary because X/Open recognises that the Transparent File Access (TFA) standardisation work is ongoing within the IEEE P1003.1f project, and X/Open intends to be compliant with P1003.1f TFA when it becomes an IEEE standard. XNFS is considered complete because it encompasses both protocols for interoperability (via the **XNFS** specification) and interfaces for application/user portability (via the **XSI** specification and the semantic differences defined in the appendices of the **XNFS** specification).

XNFS comprises a number of specifications, namely:

External Data Representation (XDR)
: This defines a syntax for describing data formats and data encoding (analogous but not equivalent to ASN.1). XDR is used to specify the other XNFS protocols.

Remote Procedure Call Protocol (RPC)
: This provides a mechanism to allow a client to call a procedure to be executed on a remote server.

Network File System (NFS)
: The X/Open specification for file-sharing services based on the NFS architecture developed by Sun Microsystems Inc.

Portmap
: A service that maps RPC program and version numbers to transport-specific port numbers thus providing a dynamic binding capability for remote programs.

Mount
: A service that looks up server pathnames, validates user identities and checks access permissions. It then provides the first file handle to clients, which then allows them entry to a remote file system.

Network Status Monitor (NSM)
: A service that provides applications with information on the status of network hosts. It is used by NLM to track hosts that have established and hold locks.

Network Lock Manager (NLM)
: An RPC-based service that provides advisory X/Open CAE file and record locking and DOS compatible file sharing and locking in an XNFS environment.

5.10.2 Internationalisation Implications

There are two types of potential Internationalisation implication in XNFS: those that are protocol related and those that arise out of Transparent File Access (TFA). The former correspond to the issues of Specification Portability and Implementation Interworking and the latter to the issues of Application Portability and Interworking.

Protocol Issues

The point at issue here is whether the encoding of parameters is language-dependent. The XNFS set of protocols uses a variety of parameter types that are clearly language-independent (for example, boolean and integers) but it also uses *string* parameters. Strings are used to specify program names, path names, and user names. However, the specification treats these as octets and does not process them, other than to transmit, store, retrieve and compare them for equality. Consequently, as far as the protocol implementation is concerned, string parameters are language-independent, and protocol implementations are both portable and, when implemented on systems with different cultural environment, will interwork. The only proviso is that strings consist of 8-bit octets.

Where the XNFS protocols (such as RPC) are used by other programs, then the unrestricted use of RPC protocols may gives rise to internationalisation problems. These problems cannot be easily solved through the use of the current internationalisation features of the X/Open CAE since the code will be running on one machine (and have access to the machine's language dependent features) but will have to code user names, path names etc. using the language-dependent features of another machine. These issues are being addressed by the Joint X/Open-Uniforum Internationalisation Group, as discussed in Section 2.5.2 on page 12 of this technical study.

TFA Problems

The purpose of XNFS is to provide transparent file access. This gives rise to issues that are, potentially, much more serious than those associated with the XNFS protocols. These issues are identified in the referenced **DISS Issue 1** X/Open snapshot; they derive from the fact that client and server do not necessarily have a common locale.

There is a similar problem with the FTAM initiator and responder for XFTAM and BSFT, but the situation is worse for XNFS. With FTAM, the codeset in use is known to both the initiator and the responder, it is only case conversion rules, collating sequences etc. that are not known, and these are only required by a minority of applications. With XNFS, not even the codeset is known.

The following example illustrates the problem. Suppose that a French user creates a file *données* on his system, which supports the codeset of ISO 8859-1. His system is networked to an English system that does not support this codeset, but simply ignores the most significant bit of any character code. The two systems are running XNFS. A directory listing of the French system, requested on the English one, displays the filename as *donnies*, the code (hex) E9 for the letter e-with-acute having been displayed as though it were (hex) 69. However, attempts to access the file *donnies* are unsuccessful, because the English system transmits code (hex) 69 to the French one, rather than code (hex) E9.

This problem is pointed out in the XNFS Specification. It can be addressed by enabling the user to establish identical locales on all the systems that he uses, and by ensuring that the same locale is used by all co-operating applications in a distributed operation. The user in the above example could then establish a French locale on the English system and work in it, using files located on his French system, as though he were working on his French system.

The establishment of identical locales on systems supplied by different vendors implies that locales must be standardised. Use of the same locale by co-operating distributed systems implies that there must be some means of conveying locale information between those systems.

A further issue is that a user application may not be aware of the locale associated with a remote file. This issue is not confined to distributed operations, as the same situation may arise with an application accessing a local file, but it is more likely to arise in a distributed system. It raises the question of whether locale information could, or should, be associated with files.

5.11 The (PC)NFS Specification

5.11.1 Overview

The X/Open **(PC)NFS** specification covers one of the two protocols sets that X/Open defines in order to provide interoperability over Local Area Networks (LANs) between personal computers and X/Open-compliant systems. (The other protocol, SMB, is discussed in Section 5.12 on page 40 of this technical study).

The **(PC)NFS** specification covers all of the protocols specified in the **XNFS** specification with the following exceptions:

- the **Network Status Monitor (NSM)** service is not used
- the **Network Lock Manager (NLM)** service is extended to provide additional services
- a new service, **PCNFSD**, is used.

With the exceptions listed above, (PC)NFS reproduces, with editorial changes, the XNFS specifications rather than referencing them.

The added facilities are as follows:

Network Lock Manager (NLM)
This is an RPC-based service that provides advisory X/Open CAE file and record locking, and DOS-compatible file sharing and locking in an XNFS environment. The (PC)NFS definition of NLM defines an upwardly compatible NLM specification, which is defined to include all the facilities of the XNFS definition and adds to it support for personal computers (namely non-monitored locks and DOS-compatible file sharing). The extra calls include parameters that specify file caller names.

Personal Computer NFS Daemon (PCNFSD)
This is a Unix daemon that provides user authentication and print services to single-user personal computer systems. (PC)NFS includes parameters to define passwords, host names, printer names, user names, filenames and spool options. The specification includes the definition of specific English characters for spooler options. However, since these are fixed (and hence can be treated by programs as arbitrary constants), this is not an internationalisation issue.

5.11.2 Internationalisation Implications

The discussion with regard to internationalisation of the **XNFS** protocol specifications also applies to the equivalent (PC)NFS versions, with the extra complication that PCs handle national language-dependent features in a different manner to X/Open compliant systems. In particular, different encodings of characters will be used between PCs and X/Open compliant systems, and PCs can change their character sets (code pages) without this becoming known to the X/Open compliant systems to which the PC is connected.

Strings, such as machine names, program identifiers and procedure identifiers will be expressed by PCs according to the DOS code page they are currently using. This will require 8-bit character support and a character encoded in the same manner may be displayed differently to users on Unix and PC systems.

The same sort of problems will arise as were illustrated in the example in Section 5.10 on page 36, but will be less easy to solve because the code page mechanism used by the PC clients is different from the locale mechanism used on the X/Open compliant servers and the two mechanisms may (for example) use different codesets for the same natural language.

5.12 The SMB Protocols Specification

5.12.1 Overview

SMB is a protocol that can be implemented as an *upper layer* over any protocol that provides the NetBIOS service. The X/Open **SMB** specification describes the SMB protocol itself and its use of NetBIOS. It also describes an OSI protocol stack and an IPS protocol stack that provide the NetBIOS service (it does this by reproducing a MAP/TOP Users' Group Technical Report and RFCs 1001 and 1002).

5.12.2 Internationalisation Implications

System Names, Resource Names and Passwords

SMB host names are upper-case characters padded with blanks. SMB names for resources (files, paths, mailslots, pipes, volumes, devices, etc.) and also passwords are encoded as null terminated ASCII strings. Pathnames may or may not be case-sensitive, and case conversion of filenames and path names is performed under some circumstances. The character set and encoding used is assumed to be ASCII and certain encodings — those for asterisk, question mark and back-slash and other special characters, and those less than (hex) 20 — have reserved meanings. Encodings greater than or equal to (hex) 80 need not be supported, and case conversion does not apply to them in any case.

This clearly impacts on specification portability. It would for example be difficult in some language and cultural environments to write an electronic mail application in which a user could freely specify mailbox names.

Some of the language and cultural dependencies could be removed by allowing any character encoding scheme that is an extension of ASCII. This would, however, still leave case conversion as a problem, and would not allow use of UNICODE/ISO 10646.

On PCs, the algorithm for case conversion is likely to be determined by the code page in use. For case conversion to be meaningful with an extension of the ASCII encoding scheme, the X/Open compliant servers would have to be aware of the code pages in use on the PC clients. This would require an extension to the SMB protocol. The X/Open compliant servers would also need knowledge of the individual code pages used on the PCs. How this could be achieved requires further study.

Data

SMB is capable of carrying 8-bit binary data. However, in section 9.1, SMBsplopen has smb_mode which can specify text mode, allowing the server to expand ASCII tabs to spaces. This options could have a slight impact on applications portability. If writing an internationally portable application that used SMBsplopen to create a spool file, the programmer would have to be careful to use graphics rather than text mode. This represents a restriction on the range of tools available to the programmer, however, rather than a restriction on what can be achieved using those tools.

Management Transactions

In section B.4.1, the parameter descriptor string, the data descriptor string and the auxiliary data descriptor are null-terminated ASCII strings. These follow a particular coded format and so are *international*. However, the type of argument passed can include a null-terminated ASCII string but not other types of character string.

This impacts slightly on applications portability. If writing an internationally portable management application, the programmer must be careful to avoid using parameters that are null-terminated ASCII strings. Again, the programmer is not restricted in what can be achieved, since it is possible to encode text in parameters in other ways (and, for an internationally portable application, it may be better in any case to avoid the use of text altogether).

5.13 The IPC Mechanisms for SMB Specification

5.13.1 Overview

This X/Open specification defines an API to the Server Message Block (SMB) protocol for use on X/Open compliant systems, provides information on the mapping of the API to SMB protocol elements and specifies the protocol elements that are required.

5.13.2 Internationalisation Implications

As regards the data passed over the API and transported using the SMB protocol, there are no limitations that affect internationalisation. The names of resources (mailslots, pipes etc.) are however required to be null-terminated ASCII strings and some comment strings transferred by the protocol are also null-terminated ASCII strings.

This clearly impacts on specification portability, as discussed in Section 5.12 on page 40.

Chapter 6

Conclusions and Recommendations

This chapter summarises the implications of internationalisation requirements on X/Open interworking specifications and presents conclusions and recommendations.

From the point of view of internationalisation, applications programs can be divided into three classes:

- an application that requires a single, fixed locale can be classed as a *single-locale* application
- an application can be classed as a *variable-locale* application if, at any time, it only processes data from a single locale, but that locale can vary from time to time
- an application that processes data from several different locales at the same time can be classed as a *multi-locale* application.

For single-locale applications, the only question is whether there is any restriction on the locales in which it can be implemented. Instances where an X/Open interworking specification imposes a restriction, or allows an implementation to impose a restriction, are identified in this technical study as issues of specification portability. Except for some issues arising from the use of specifications that X/Open can not change unilaterally, the only restriction now imposed by X/Open interworking specifications is that some API functions can not be used in conjunction with UNICODE or ISO 10646 codesets, because the functions rely on strings being null-terminated.

The question of UNICODE/ISO 10646 has been discussed by X/Open working groups. There has been a reluctance to replace function arguments of type **char** with arguments of type **wchar_t**. However, X/Open will now consider defining additional functions that perform the same actions as the API functions that rely on strings being null-terminated, but that use arrays of type **wchar_t** rather than arrays of type **char**. These functions should not replace the existing functions, but should provide an alternative to them for use by internationalised applications. The functions affected are the XTI functions *t_error*( ) and *t_strerror*( ), and the XAP functions *ap_get_env*( ) (which can return an **ap_diag_t** value) and *ap_error*( ).

For variable-locale applications, there are further questions that arise. These are:

- whether the specification requires the implementation to allow variable-locale applications to be written
- whether the specification requires the implementation to provide explicit support for variable-locale applications, for example by tagging information with locale identifiers.

Except for some cases where X/Open can not change the specifications unilaterally (identified as issues of applications portability), the X/Open interworking specifications considered in this technical study do require implementations to allow variable-locale applications to be written. However, they do not require implementations to provide explicit support for variable-locale applications. This means that interworking applications must make explicit provision for variable-locale operation where this is required.

The situation for multi-locale applications is similar. Except in some cases where X/Open can not change the specifications unilaterally, the X/Open Interworking Specifications considered in this technical study do not contain features that could be implemented in a way that would prevent multi-locale applications from being written, but they do not require implementations to provide explicit support for such implementations. Interworking applications must therefore make explicit provision for multi-locale operation where this is required.

In general, the X/Open interworking specifications:

- can be used by single locale applications that do not use codesets (such as UNICODE or ISO 10646) that allow embedded nulls (and changes that would enable such codesets to be used will be considered)

  and

- can be used by variable-locale and multi-locale applications (with the same restriction on the codesets that they can use)

  but

- do not provide explicit support (for example, by attaching locale identifiers to information) for variable-locale and multi-locale applications.

The exceptions to this (assuming that the change described in Chapter 7 of this technical study is incorporated in the **XTI** specification) are listed below:

- XFTAM provides non-locale-dependent diagnostic strings (but this is in accordance with the FTAM standard)
- XBSFT has an English-language user interface
- some of the OM Attributes of the XOM, XFTAM, BSFT, X.400, XMS and XDS APIs represent character strings, but the ASN.1 string syntaxes that can be used are not sufficiently general to cater for all possible codesets (but this is in accordance with the underlying OSI standards)
- the Directory to which the XDS provides an interface carries out character string comparisons that are not locale-dependent (but this is in accordance with the underlying X.500 standards)
- NetBIOS and SMB implicitly assume an ASCII codeset and this has implications for the XTI, SMB and IPC for SMB specifications.

Chapter 7

Change Requests for Internationalisation

This chapter contains the formal change requests for the X/Open interworking specifications that were originally raised in the previously published X/Open **Interworking Internationalisation** snapshot. This chapter also describes how each of these change requests has since been resolved.

These change requests address the issues that could be addressed by X/Open in isolation or in co-operation with the X.400 API Association. They do not address issues requiring modifications to international standards or consultation with the PC supplier community.

7.1 The XTI Specification

The following changes to the **XTI** specification were proposed.

Document: The **XTI** Specification
X/Open CAE Specification, C196 or XO/CAE/91/600 (January 1992).

Change number: XOP-1

Source: X/Open / I18n CR from I18n of Interworking Specs SS

Title: Internationalised Error Messages
Functional Upgrade

Qualifier: Major Technical

Rationale: The functions *t_error*( ) and *t_strerror*( ) return strings. For international operation, these should be in the natural language of the locale established by the application program.

Change:

i. In the man page for *t_error*( ), change the description as follows:

   a. Remove the text ‘‘language-dependent’’ from the first line.

   b. In the first line of the third paragraph, replace the text

      ‘‘implementation-defined.’’

      with

      ‘‘dependent on the current locale.’’

ii. In the man page for *t_strerror*( ), replace the text in the description:

   ‘‘language-dependent error message string’’

   with

   ‘‘message string based on the current locale’’

   Also, replace the text:

   ‘‘implementation-defined’’

   with

   ‘‘the natural language based on the current locale’’

Document: The **XTI** Specification
X/Open CAE Specification, C196 or XO/CAE/91/600 (January 1992).

Change number: XOP-2

Source: X/Open / I18n CR from I18n of Interworking Specs SS

Title: Internationalised Error Messages
Functional Upgrade

Qualifier: Major Technical

Rationale: Some XTI functions return pointers and character strings as arguments to function calls or the return value from a function call. For internationalised operation, this should be in the natural language of the locale established by the application program.

Change:

i. In the man page for *t_error*( ), replace:

```
char *errmsg
```

with:

```
wchar_t *errmsg
```

ii. In the man page for *t_strerror*( ), replace:

```
char *t_strerror
```

with:

```
void *t_strerror
```

Change Request (CR) XOP-1 was accepted (with a minor modification to the wording). It has not yet been implemented, but will be implemented in the next published version of the **XTI** specification.

CR XOP-2 was withdrawn, for reasons discussed in Chapter 6, and X/Open will now consider the addition of parallel interfaces using **wchar_t** rather than simply replacing existing interfaces that use **char**.

7.2 The XAP, XAP-TP and XAP-ROSE Specifications

The following changes to the **XAP** specification were proposed:

Document: The **XAP** Specification
X/Open Preliminary Specification, P203 (June 1992).

Change number: XOP-3

Title: Internationalised Diagnostic Messages

Qualifier: Minor Technical

Rationale: Function *ap_get_env*( ) returns a diagnostic message in field error of structure *ap_diag_t* when passed the AP_DIAGNOSTIC attribute. For internationalised operation, this should be in the natural language of the locale established by the application program. To enable the character set of any natural language to be used, and to allow for the encoding scheme of ISO 10646, a null-terminated character string should not be used.

Change:

i. In the ENVIRONMENT man pages definition of type *ap_diag_t*, (page 54) and in the repeat definition in Appendix A (page 195), change:

```
char *error      /* textual message */
```

to:

```
wchar_t *error   /* textual message */
```

ii. Change the ENVIRONMENT man pages description of the error field (near the bottom of page 55) to:

‘‘The error field will be set up to point to a text string, in the natural language of the current locale, which describes the error condition, or will consist of a single null character if no such text string is available.’’

Document:	The **XAP** Specification X/Open Preliminary Specification, P203 (June 1992).
Change number:	XOP-4
Title:	Internationalised Error Messages
Qualifier:	Minor Technical
Rationale:	Function *ap_error*( ) returns an error message. For internationalised operation, this should be in the natural language of the locale established by the application program. To enable the character set of any natural language to be used, and to allow for the encoding scheme of ISO 10646, a null-terminated character string should not be used.
Change:	In the *ap_error*( ) man page:

i. change:

```
char *ap_error  (aperrno)
```

to:

```
wchar_t *ap_error  (aperrno)
```

ii. change the DESCRIPTION to"

‘‘This function returns a pointer to a message, in the natural language of the current locale, that describes the error indicated by aperrno. The pointer shall point to a null character if no such message is available. For English language locales, the messages shall be those that are listed in the XAP interface functions introductory manual pages in this specification.’’

Part b) of CR XOP-3 and CR XOP-4 were accepted and have been implemented in the referenced **XAP** X/Open CAE Specification.

Part a) of CR XOP-3 and CR XOP-4 were not accepted. As discussed in Chapter 6, X/Open will now consider the addition of parallel interfaces using **wchar_t**, rather than simply replacing existing interfaces that use **char**.

7.3 The XOM Specification

The following change to the **XOM** specification was proposed:

Document: The **XOM** Specification
X/Open CAE Specification, C180 or XO/CAE/91/080 (November 1991).

Change number: XOP-5

Title: Support for Universal and Unrestricted Character Strings

Qualifier: Major Technical

Rationale: The character string types supported by XOM are not sufficiently general to support all character sets and codesets used internationally. This issue has been recognised by the standards bodies responsible for ASN.1. A new DIS 8824 is currently in ballot. It includes (among other changes from the present version) support for two new string types. These are:

Universal String — which supports the Universal Multi-Octet Coded Character Set of ISO DIS 10646

Unrestricted String — which supports any character set and encoding scheme, provided they have OIDs.

These string types are sufficiently general for full international use. It would be possible to make the changes to XOM that would add support for these types independently of any other changes that might be suggested to reflect other changes that have been made to ASN.1

Change:

i. In Section 3.4, Table 2, add, after ''UTC Time'':

''Universal[2] ''

''Unrestricted[4] ''

and add the following footnote:

'' [4] Values of this syntax are represented in their BER encoded form.''

ii. In Section 3.8, Table 5, add, after the ''Teletex String'' line:

''Universal String String(Universal)''
''Unrestricted String String(Unrestricted)''

iii. In Section 4.2.12, replace:

''A zero character follows''

with:

''Universal and Unrestricted strings can contain null octets. Only the length-specified form shall be used to represent strings of these types. When either form is used, a null character (that is, an instance of the character whose encoding is zero) follows''

In Section 4.2.13, add to clause 4, after:

''teletex string''

the following:

‘‘universal string, unrestricted string’’.

iv. In Section 4.5, add to the /* Syntax */ section, after the definition of OM_S_TELETEX_STRING, the following:

```
#define OM_S_UNIVERSAL_STRING     ((OM_syntax)28)
#define OM_S_UNRESTRICTED_STRING  ((OM_syntax)29)
```

Change Request (CR) XOP-5 was accepted and has been implemented (with minor changes) in the referenced **XOM** X/Open CAE specification.

7.4 The BSFT Specification

The following change to the **BSFT** specification was proposed:

Document: The **BSFT** Specification
X/Open CAE Specification, C194 (December 1991).

Change number: XOP-6

Title: International Support for User Interfaces

Qualifier: Major Technical

Rationale: The user interface for BSFT is defined to use only the English language. This restricts the use of BSFT internationally.

Change: In Appendix A, immediately after the first paragraph of the Appendix ("This section defines the user interface for the BSFT facility."), add the following paragraphs:

‘‘This section specifically defines an English language user interface. An implementation may, however, support user interfaces in other languages.’’

‘‘In each user interface supported, other than the one specifically defined in this section, there shall be commands, parameters and responses that are equivalent to all those that are defined in this section. How the commands, parameters and responses of each user interface correspond to those defined in this section shall be documented.’’

‘‘If an implementation supports more than one user interface, and one of the environment variables LC_ALL, LC_MESSAGES, or LANG is set when the BSFT facility is invoked, and a user interface appropriate to the locale referenced by that environment variable is supported, then that user interface shall apply. If more than one of those environment variables are set, then the order of precedence shall be LC_ALL first, LC_MESSAGES second, and LANG third.’’

‘‘It should be noted that a user who attempts to transfer a file that has been defined using language and cultural conventions other than those of his current locale may experience problems, and that the BSFT facility does not provide a means for the user to determine the locale in which a file (on either the local or the remote system) has been defined, or to determine or influence the locale assumed by the BSFT responder.’’

This CR was rejected, with the rationale that internationalisation of the BSFT user interface should be addressed as part of the solution of the larger issue of internationalising the X/Open commands.

7.5 The XFTAM Specification

The following change to the **XFTAM** specification was proposed:

Document: The **XFTAM** Specification
X/Open Preliminary Specification, P206 (September 1992).

Change number: XOP-7

Title: Internationalised Error Messages

Qualifier: Minor Technical

Rationale: Function *gperror*( ) returns two strings. For internationalised operation, these should be in the natural language of the locale established by the application program. To enable the character set of any natural language to be used, the requirement that they must be printable strings should be dropped. (The proposed change would allow any character string type representable in XOM. In conjunction with CR XOP-5, proposed for XOM, this would allow Universal and Unrestricted Strings).

Change: In the man page for *ft_gperror*( ),

i. In the description of Return_string, change:

‘‘Return_string(OM_String(Printable))
The printable string ... NULL-terminated.’’

to:

‘‘Return_string(OM_String(*))
A text string in the natural language of the current locale that represents the Return-Code attribute of the API_out_in parameter. This is the XFTAM-specified error code and is always returned. The resulting string is formatted for printing as a self-contained unit (for example, in an English language locale, it includes a terminating newline character).’’

ii. In the description of Vendor_string, change:

‘‘Vendor_string(OM_String(Printable))
The printable string ... NULL-terminated.’’

to:

‘‘Vendor_string(OM_String(*))
A text string in the natural language of the current locale that represents the Vendor-Code attribute of the API_out_in parameter. This is an optional implementation-specific error code and shall not be returned if the API_out_in parameter did not contain an equivalent code. The resulting string is formatted for printing as a self-contained unit (for example, in an English language locale, it includes a terminating newline character).’’

CR XOP-7 was accepted and has been implemented in the referenced **XFTAM** X/Open CAE Specification.

7.6 The X.400 API Specification

The following change to the **X.400 API** was proposed:

Document: The **X.400 API** Specification
X/Open CAE Specification, C191 or XO/CAE/91/100 (December 1991).

Change number: XOP-8

Title: Printable Strings Used Internationally

Qualifier: Minor Technical

Rationale: Following the 1988 version of the X.400 Series Recommendations, the **X.400 API** requires use of Printable Strings for certain O/R Address attributes when the API is used to send messages internationally. The corresponding requirement on the protocol was dropped from ISO 10021 and is expected to be dropped from the 1992 version of the X.400 Series Recommendations.

Change: In Section 5.2.31, delete the first and last sentences of note 3, that is:

a. delete ‘‘If only one value ... String(Printable).’’

b. delete ‘‘Printable strings are required internationally ... communication.’’

CR XOP-8 was accepted and has been implemented in the referenced **X.400 API, Issue 2** X/Open CAE Specification.

7.7 The XDS Specification

The following change to the **XDS** API was proposed:

Document:	The **XDS** Specification X / Open CAE Specification, C190 or XO / CAE / 91 / 090 (November 1991).
Change number:	XOP-9
Title:	Support for Directory Strings
Qualifier:	Major Technical
Rationale:	Following X.520 (1988), XDS defines a number of Directory Attributes to have syntax String(Teletex). This is not sufficiently general to support all character sets and codesets used internationally. This issue has been recognised by the standards bodies responsible for X.520. A new version of X.520 has been proposed and (in this respect at least) has been agreed from a technical point of view, but has not yet been formally adopted by the CCITT. This introduces a new syntax, "Directory String", for these attributes. Assuming that CR XOP-5 is accepted for XOM, it would be possible to make the changes to XDS that would add support for this syntax, and to make them independently of any other changes that might be suggested to reflect other changes that are made to the X.500 Series recommendations.

Change:

i. In Section 1.1, add, after ‘‘1988’’, a superscript 1 and place the following footnote at the bottom of the page:

   ‘‘[1] It also takes account of some changes made in the 1992 version of these standards.’’

ii. In Section 7.2, add, after:

   ‘‘C constants start with DS_A.)’’

   a new paragraph:

   ‘‘Several of the attribute types are defined in the 1992 version of the Standards to have ASN.1 syntax DirectoryString. This is a CHOICE of TeletexString, PrintableString and UniversalString. In these cases, the values of the corresponding Attribute-Values OM attributes can have syntaxes String(Teletex), String(Printable) or String(Universal). This is indicated by describing their syntaxes as String(Directory).’’

iii. In Section 7.2, 4th paragraph:

   a. Change

      ‘‘two general rules’’ to ‘‘three general rules’’

   b. Add, at the end of the paragraph:

      ‘‘For all attribute values whose syntax is indicated as String(Directory), differences in the case of alphabetical characters shall be considered insignificant and, if the strings being compared are of different syntax, the comparison shall proceed as normal so long as the corresponding characters are in both character sets, but shall fail otherwise.’’

iv. In Section 7.2, Table 34, in the entries for A-Business-Category, A-Common-name, A-Description, A-Knowledge-Information, A-Locality-Name, A-Organisation-Name, A-Organisational-Unit-Name, A-Physical-Delivery-Office-Name, A-Post-Office-Box, A-Postal-Code, A-State-Or-Province-Name, A-Street-Address, A-Surname, A-Title, change:

‘‘String(Teletex)’’

to:

‘‘String(Directory)’’

(This should remove all occurrences of String(Teletex)" from the table.)

v. In Section 7.12, change the Value Syntax of Postal-Address from:

‘‘String(Teletex)’’

to:

‘‘String(Directory)’’.

Note: Since Teletex String is a special case of Directory String, the uses of DS_A_COMMON_NAME etc. In the programming examples of **XDS** Chapter 9 need not be changed.

CR XOP-9 was accepted and has been implemented in the referenced **XDS, Issue 2** X/Open CAE Specification.

7.8 The XNFS Specification

No change requests were proposed for this specification.

7.9 The (PC)NFS Specification

No change requests were proposed for this specification.

7.10 The SMB Protocols Specification

No change requests were proposed for this specification.

7.11 The IPC Mechanisms for SMB Specification

No change requests were proposed for this specification.

X/Open Technical Study

Part 3

X/Open Data Management Specifications

X/Open Company Ltd.

Chapter 8

Introduction

The chapters in this part of this technical study consider the impact of internationalisation on the X/Open data management specifications. These specifications are at present either CAE specifications or are expected shortly to become preliminary specifications. They consist of the following documents (full details are given in **Referenced Documents** on page xv).

- the **SQL** specification
- the **CLI** specification
- the **RDA** specification.

Structure of This Part

Chapter 9 discusses general internationalisation issues that are associated with SQL, as defined by the ISO standards. Many of these issues are also common to the X/Open data management specifications.

Chapter 10 examines the implications of internationalisation on the data management specifications listed above.

After this, Chapter 11 presents conclusions and recommendations.

Chapter 9

Structured Query Language (SQL)

9.1 Overview

The X/Open data management specifications (**SQL**, **CLI** and **RDA**) are all related to Structured Query Language (SQL). This chapter discusses the internationalisation issues associated with SQL in general. It is a prelude to Chapter 10 which discusses the X/Open data management specifications individually.

SQL has been standardised by ISO. The most recently approved SQL standard is ISO/IEC 9075: 1992; this is sometimes called SQL 92 or SQL2 (see the referenced ISO SQL 92 standard). This technical study uses the term SQL 92 when referring to this standard. The standard specifies the syntax and semantics of SQL, for use where SQL statements are used in the following ways:

- invoked directly (for example from a user interface)
- stored as procedures that can be called from programs
- embedded in programs written in other programming languages.

For direct invocation, SQL 92 does not define how the statements are invoked or how any results are returned. For calling SQL procedures from, or embedding SQL statements in, a programming language, SQL 92 includes syntax for the particular programming languages Ada, C, COBOL, Fortran, MUMPS, Pascal and PL/I.

SQL 92 defines three levels of conformance:

- Entry Level SQL
- Intermediate SQL
- Full SQL.

Intermediate SQL is a subset of Full SQL, and Entry Level SQL is a subset of Intermediate SQL.

Work is currently proceeding on a new version of the SQL standard, currently referred to as SQL3. This is documented in the referenced ISO SQL3 draft standard, which incorporates the following significant new features:

- active ‘‘rules’’, called triggers
- abstract data types
- multiple null states
- PENDANT referential integrity
- a recursive union operation for query expressions
- enumerated and boolean data types
- SENSITIVE cursors.

The SQL3 draft standard does not include any significant features related to internationalisation beyond those already contained in Full SQL 92.

9.2 Multiple Character Sets

Character strings constitute one of the types of data that can be stored in relational databases and manipulated using SQL. SQL provides means for entering and retrieving character string data, and for sorting and ordering character string data.

Clearly, users may wish to use character string data that relates to any language and cultural environment. They may even wish to store, in the same table, character string data that relates to several different language and cultural environments. For example, a user might wish to use a table to store a multi-lingual glossary, with columns containing equivalent terms and definitions in English, French, Russian, Arabic and Japanese. The same user might then wish to derive an appropriately sorted copy of the table for each of these languages.

If users are to be able to carry out such operations, SQL must allow:

- the use of the appropriate character sets
- several different character sets to be used in a single table
- the sorting operation to use the collation order appropriate to the character set or sets of the data to be sorted.

Generally, these conditions are satisfied by Full SQL (as defined in SQL 92), which is rich in features relating to international use. The conditions are however only partly satisfied by Intermediate SQL, and they are not satisfied at all by Entry Level SQL. In particular:

- Full SQL and Intermediate SQL, but not Entry Level SQL, allow *character set specifications* that allow an application to use several different sets of characters.
- Full SQL, but not Intermediate SQL or Entry Level SQL, also allows *collation definitions*, which allow an application to specify collating sequences other than the default collating sequence for a character set.

9.3 Use of Standard Names

Although Full SQL has features that allow an application to specify multiple character sets and collating sequences, it does not require that the implementation supports standard identifiers for them. So, for example, one implementation might support a character set called **ISO8859-1**, while another might support the same character set but call it **ISO1**. This clearly inhibits portability of applications.

This problem would be overcome if there were a set of standard character set identifiers which all SQL implementations must use. It would be advantageous if these identifiers were the ones defined for locales in the registry proposed in the **DISS Issue 1** snapshot. An application could then make SQL statements such as:

```
CREATE TABLE foo (
     col_a CHARACTER (10) CHARACTER SET ge_GE.8859-1
                          COLLATE ge_GE.8859-1@foobar,
     col_b CHARACTER (20) COLLATE fr_FR.8859-1
)
```

and be assured of portability across a wide range of implementations.

Failing the definition of standard names for the whole user community, functional profiles and corporate or national procurement standards may define their own sets of names to ensure portability. For example, the FIPS 127-2 standard defines the character set names LATIN1, ASCII_FULL, and ASCII_GRAPHIC.

9.4 Character Set Not Determined by Locale

In order to write a fully internationalised application — that is, an application that exploits all the features of SQL and which can be used without re-compilation in any language and cultural environment — it is necessary that the character set assumed by the SQL processor in direct execution and dynamic execution of SQL statements can be determined by the locale mechanism.

SQL 92 does not refer to locale mechanisms. Instead it provides the SET NAMES statement which allows the application programmer to nominate the default character set for dynamic SQL statements and for direct invocation of SQL statements. This statement is only available in Full SQL.

An application could thus:

- determine the codeset of the current locale by calling *nl_langinfo*(CODESET)
- use the SQL SET NAMES statement to set the default character set names for identifiers and character string literals in preparable statements.

The problem is that the application must be able to convert the codeset name obtained by means of *nl_langinfo*( ) into the character set name required by SQL. This cannot be done in a portable way (except in the context of a local standard for such names, such as that defined in FIPS 127-2), since neither **XPG4** nor the SQL 92 standard defines standard character set names.

An internationalised program that uses locales may make use of the features of Full SQL to run in different language and cultural environments, but is not able to achieve full internationalisation in a portable way. The character set and collation features are however not available in Entry Level SQL, and only some of them are available in Intermediate SQL. It is hard to see how an application can have any significant degree of internationalisation if only the facilities of Entry Level SQL are available.

Multiple character set facilities (as in Full SQL) and locales (as in **XPG4**) are not the only ways in which internationalisation could be provided. For example, each item of data could be *self-announcing*, that is, could contain an indication of the language and cultural environment in which it should be processed. A mechanism to support this is described in the **DISS Issue 1** snapshot. Such a mechanism could be used in conjunction with SQL — even with Entry Level SQL. The use of such a mechanism in conjunction with SQL forms no part of current SQL standardisation work, however.

9.5 Encodings

SQL 92 does not specify how character (or other) data is represented internally by an SQL implementation, nor does it say how data is represented when passed between the SQL implementation and other system components. However, SQL 92 does attach semantics to the data. For example, each item of character data is understood to consist of a set of characters from some known character set. Clearly, other components of a system that includes SQL must attach the same semantics to the data as the SQL component. This means that the components must all use the same character set encodings.

Direct Invocation

In direct invocation, the manner in which the statements are invoked and the results are returned (and hence the manner in which data is passed) is implementation-defined. This means that vendors must document the character sets and encodings (the *codesets*) that they can accept, so that the SQL implementation can be used in conjunction with other products.

Non-direct Invocation

In other cases, the system components that pass data to or receive data from an SQL implementation are typically:

- the text editors used to create SQL statements
- the host language processors (including compilers, interpreters, and run-time libraries) that process host language programs that call SQL procedures, or in which SQL statements are embedded.

Integration with Text Editors

The issue of text editors (the term is used here to include text processing programs of all types) arises when an SQL application refers to particular character strings. Such strings may be used as names (table names, column names and so on), or as application data (for example, in WHERE clauses of SELECT statements). If such strings are to be meaningful, the SQL implementation must use the same encoding scheme as the text editor. Literal character strings are likely to be specific to a particular language, however, and their use for applications data should therefore be avoided in internationalised applications. This might be done, for example, by using character string variables whose values are obtained from message catalogues.

Integration with Host Language Processors

Data can be passed between SQL and a host language processor in several ways: for example, in the parameters of SQL procedures, or in variables of embedded SQL statements.

A programmer can hard code character encodings into a program, for example, with the C statement:

```
c = (char) 65;
```

The use of such constructs should however be avoided when writing internationalised software, as discussed in the referenced **Internationalisation Guide**. If the advice given in the **Internationalisation Guide** is followed, the character encodings are determined entirely by the host language processor and the currently established locale.

Although SQL 92 requires an implementation to state which character sets it can handle, SQL 92 does not clearly state a requirement for implementations to describe the encodings that are used to represent the characters of those character sets. For example, an implementation could support an English character set, without saying whether it is encoded using ASCII, EBCDIC, UNICODE, or some other encoding scheme. This information is needed to determine whether a particular implementation of SQL and a particular host language processor can interwork. It is therefore important that an SQL implementation should specify the character set encodings that may be used by a host language processor for procedure parameters, embedded variables and executable SQL statements.

9.6 String Operations

The use of encodings such as UNICODE and ISO 10646 in which a single graphic symbol may be represented by a combination of several members of the codeset raises questions with regard to string comparisons and related operations. These concern whether a graphic symbol represented by a combination of several codeset elements should be treated as a single character or as multiple characters for collation purposes. SQL 92 appears to treat such a combination as several characters rather than one[5].

Although the use of combining characters in UNICODE is a particularly topical instance of this issue, it also arises for other character set encodings. Spanish, for example, requires the character **ch** to collate as a single entity. This character does not have a single codeset element in ISO Latin-1, and the collation mechanism must take account of this.

As far as collation operations are concerned, collating sequences are regarded in SQL 92 as operating on strings rather than on individual characters. However, the default collating sequences for standard character repertoires and standard universal forms of use are based on the numerical order of codeset elements; they process strings character by character. Also, such orders have no intended relationship to natural language collation orders, so they may not produce the desired ordering. This may lead to counter-intuitive results in some cases. For example, a string containing lower case <e> followed by an acute accent combining character would collate differently from one that contains a single <e with acute accent> character, and neither collation would be that required by French, where <e with acute accent> is generally treated for collation purposes as though it were the simple letter <e>.

Counter-intuitive results may also be obtained from operations involving wildcard characters and from those in which substrings are indicated by their numeric positions within strings. For example, the second letter in the French string "défense de fumer" is <e with acute>. Is <f> the third or the fourth character of that string?

9.7 ISO 10646 and C

The introduction of UNICODE or ISO 10646 encodings raises questions concerning the use of SQL in conjunction with a C host program. An SQL implementation based on UNICODE or ISO-10646 might use 2-byte or 4-byte character encodings, but these cannot conveniently be handled as character strings by a C host program. Moreover, SQL character data is represented in C by null-terminated strings, while UNICODE and ISO 10646 allow character encodings that include null bytes, as discussed in Section 2.3 on page 9. How these difficulties should be handled is not yet clear, but possibilities include:

- The C program encodes the characters using an encoding that is legal for C (such as that in the ISO 8859 standard or a UTF) and the SQL implementation converts this to UNICODE / ISO 10646.

---

5. The recommendation by many members of the Unicode Consortium is to decompose the pre-combined characters (for example, A-Umlaut = A + Umlaut), then conduct sorting based on letter, case, cultural and accent weights.

- The C program stores characters as quantities of type **wchar_t**, but the SQL implementation treats them as arrays of type **char**. A future version of the SQL standard will require strings to be terminated by the number of null bytes appropriate to the encoding scheme (two null bytes or four null bytes for ISO 10646, as opposed to a single null byte for ASCII). A recent Erratum to the SQL 92 standard does in fact clarify that the two or four null bytes are required.

These difficulties do not arise with languages such as COBOL, which do not constrain the way that character strings are encoded.

9.8 Reserved Words and Special Characters

SQL 92 defines a number of reserved words (ABSOLUTE, ACTION, ADD and so on) and gives a special importance to characters such as double quote, percent and ampersand. These have an English language flavour in that the reserved words are meaningful in English. The usage of some of the special characters is similar to their usage in English (for example, the use of quote characters to begin and end a character string literal; many languages use different characters for these purposes). This is undoubtedly an inconvenience to anyone developing an SQL application in a non-English environment. However, it does not affect the degree to which an SQL application can be internationalised. It is probably acceptable, given the technical difficulties of implementing a multi-lingual or internationalised version of SQL.

9.9 Numeric and Date Literals

SQL 92 defines character string representations of numbers and dates (*numeric literals* and *date literals*) that use the period character as a decimal separator, and have a year-month-day format for dates. These are not the natural representations in all cultural environments. In an internationalised application, they would have to be converted to a locale-dependent format before being displayed at the user interface.

9.10 Diagnostic Information

The GET DIAGNOSTICS statement specified by SQL 92 can include character strings representing alpha-numeric codes (SQLSTATE) and implementation-defined character strings (MESSAGE_TEXT). The SQLSTATE strings, which are limited to using the 10 digits and the 26 upper-case Latin alphabetic characters, are a possible inconvenience to application developers in some language and cultural environments, but do not affect the degree to which applications can be internationalised. Applications that use implementation-defined MESSAGE_TEXT strings may be dependent on the language in which those strings are written. Internationalised applications should not use such strings as a source of error message text.

SQL 92 says that when MESSAGE_TEXT is requested by the application, an implementation may set the string returned to spaces, to a zero length string, or to a character string describing the condition indicated by the returned SQL_STATE value. If implementations follow this precept (which is in a non-normative note) then applications should not need to use MESSAGE_TEXT strings since the relevant information should also be given by SQL_STATE. Of course, it is possible that an implementation could return a string obtained from a message catalogue and determined by the current locale. There is no requirement on implementations to do this; an application that relied on such behaviour would not be portable.

9.11 Arithmetical Expressions

The English representation of arithmetical expressions and numbers may not be appropriate for all language and cultural environments. For example, some environments may use a notation other than the ''arabic'' one for representing numbers. This implies a requirement for the application to convert such information to the form required by the local language and cultural environment, but does not prevent an application that performs such a conversion from being internationalised. It should be noted, however, that the locale facilities currently specified by X/Open do not provide for differing arithmetical conventions.

9.12 Directionality

SQL 92 (and other SQL specifications) appear to assume a left to right and top to bottom, row-wise, display scheme. Such a scheme is not appropriate to languages (like Hebrew and Arabic) which scan right to left, or to languages that scan not only right to left but also column-wise rather than row-wise.

In fact, these issues are concerned with the way that information is presented at the user interface, rather than how it is organised in the database. A *row* can be presented in any direction: left to right, right to left, top to bottom or bottom to top. SQL 92 does not specify how information is to be presented at the user interface; furthermore none of the X/Open data management specifications are concerned with the user interface. These issues are therefore not relevant to this technical study.

Data Management Specifications

10.1 The SQL Specification

10.1.1 Overview

The X/Open **SQL** specification defines the application programming interface for X/Open-compliant relational database management systems. It includes almost all of the Entry Level provisions of SQL 92 relating to embedding SQL in C and COBOL host language programs. It also includes some features (in particular, Dynamic SQL) from the Intermediate and Full levels of SQL 92.

10.1.2 Internationalisation Implications

The Intermediate and Full level SQL features in the X/Open **SQL** specification do not include the multiple character set and collation sequence features. All of the issues identified for Entry Level SQL in Chapter 9 of this technical study therefore apply to the **SQL** specification. There are no additional issues.

10.2 The CLI Specification

10.2.1 Overview

The **CLI** specification describes an API for database access that is an alternative to the API defined in the X/Open **SQL** specification. The **SQL** specification describes how to create SQL statements that can be embedded in a C or COBOL source program and, after suitable pre-processing, can be compiled by a C or COBOL compiler. This approach is not always the best one. The **CLI** specification defines C functions and COBOL subroutines that can be called from C or COBOL programs and that will provide the functionality that is provided by the SQL Specification. It also provides functions and subroutines that enable the programmer to control connections between database clients and servers.

Note that this is a different approach from the use of procedures as defined in SQL 92. Procedures are SQL statements that can be called from a host program, whereas the CLI comprises C or COBOL routines that cause SQL statements to be executed.

10.2.2 Internationalisation Implications

General Implications

As for the **SQL** specification, all the issues identified for Entry Level SQL in Chapter 9 of this technical study apply to the **CLI** specification.

Passing of Character Strings

There is a difference between the **CLI** specification and the **SQL** specification in the way that character strings are passed between the host language processor and the SQL implementation.

For the **SQL** specification:

- character strings are passed in variables
- dynamic SQL statements are passed in variables
- other SQL statements are embedded in the host source program (and so are part of the input to the SQL pre-processor).

For the **CLI** specification:

- all SQL statements and character string data are passed as function and subroutine arguments — for example, the *vcSqlStr* argument of *prepare*( ) and the *vcColName* argument of *DescribeCol*( ).

The same considerations apply to such arguments as apply to character string variables in embedded SQL. In particular, there are the same problems with UNICODE/ISO 10646 and null terminated strings in C. Thus, this difference between the **CLI** specification and the **SQL** specification introduces no new internationalisation issues.

Character String Conversions

CLI provides for automatic conversion between numeric values and character strings. The application can supply or retrieve a numeric value in the form of a character string; the implementation performs the conversion to or from numeric format. The character string representation of numeric values is that defined for numeric literals in SQL 92. It is not the natural representation in all language and cultural environments but, as discussed in Chapter 9 of this technical study, its use does not prevent the internationalisation of applications.

10.3 The RDA Specification

10.3.1 Overview

The **RDA** specification defines the format for remote communications with an SQL database. It is based on the ISO RDA standard, and describes OSI Application Protocol Data Units (APDUs) and their use in conjunction with the Association Control Service Element (ACSE) and the Presentation and Session services.

10.3.2 Internationalisation Implications

General Implications

The **RDA** specification does not impose constraints on the types of SQL statement that can be executed remotely. The issues pertaining specifically to Intermediate and Entry Level SQL therefore do not apply.

The **RDA** specification states the following:

- SQL statement text and SQL character string data are represented as octet strings
- object identifiers for their encodings are associated with them.

The issues pertaining to locale-dependence of the character sets and to the definition of the encodings of such character data therefore do not apply to it (but see **Visible Strings** (below) for issues pertaining to the encodings of other character data).

The other issues pertaining to Full SQL apply.

Visible Strings

The **RDA** specification requires that certain information is represented by ASN.1[6] *Visible Strings*. This effectively means that the information must be encoded in basic ASCII. The information concerned includes *diagnosticInformation* (various services), *identityOfUser* (R-Initialise), *aborted* (R-Status) *dataResourceName* (R-Open), and *colName*, *classOrigin*, *subclassOrigin*, *messageText*, *sQLState, sQLErrorText* (R-ExecuteDBL).

In the case of *sQLState*, which contains formally coded information, this is an inconvenience to the application developer, but nothing worse (see the discussion of SQLSTATE in Section 9.10 on page 68 of this technical study).

In other cases, the developer of an internationalised application, or of an application in a language and cultural environment other than English, is hampered to a more serious extent. The developer may, for example, wish to display column names at the user interface of the client system. The names typically include non-ASCII characters. Therefore they cannot be passed to the server, because they cannot be encoded as Visible Strings. It would, of course, be possible to define a second set of column names that use only ASCII characters, and to translate them to the non-ASCII names before displaying them at the user interface, but this is an overhead which is undesirable.

---

6. Abstract Syntax Notation 1 (ASN.1) is a formal notation for describing information types. It is used to describe the types of information conveyed by the OSI presentation service. It is defined in the referenced ASN.1 standard.

With information such as *sQLErrorText*, which is generated by the server and returned to the client, the situation is more serious still as the application may not be able to determine the corresponding text for the user's language and cultural environment; it is therefore unable to display the information.

These issues apply not only to the **RDA** specification but also to the ISO RDA standard on which it is based.

Work is proceeding in ISO (ISO / IEC JTC1 / SC21) on the addition to ASN.1 of base types dealing with ISO 10646. Once this work is stable, it would be possible for the **RDA** specification and the ISO RDA standard to refer to it. However, the following points must be considered:

- it would mean that data that already exists and is encoded using some other character set, such as that in the ISO 8859 standard, would have to be converted to the ISO 10646 standard form
- it would not enable collating sequences, conversions, and other information pertinent to the language and cultural environment to be identified.

Chapter 11

Conclusions and Recommendations

This chapter presents conclusions and recommendations regarding the internationalisation of X/Open data management specifications.

11.1 Conclusions

The following conclusions can be drawn from the analysis of internationalisation issues in this technical study:

(C-01) The internationalisation problems associated with the X/Open data management specifications are not introduced by additions to or divergences from the related International Standards; those standards have the same problems.

(C-02) A portable application that uses an implementation of the **SQL** specification or the **CLI** specification cannot in general be fully internationalised, because the character sets are not determined by the current locale. The SQL implementation need not be aware of the currently established locale. Therefore it may not correctly interpret the character encodings presented to it. This issue is discussed in Section 9.4 on page 65.

(C-03) Internationalisation of applications that use implementations of the **RDA** specification is limited, because of the requirement that certain character data be represented by ASN.1 Visible Strings, which means that it is essentially restricted to being representable using ASCII. This issue is discussed under **Visible Strings** on page 74.

(C-04) The use of diagnostic message text (MESSAGE_TEXT) to convey information to applications is inappropriate for internationalised applications. This issue is discussed in Section 9.10 on page 68.

(C-05) The Entry Level SQL facilities upon which the **SQL** specification and the **CLI** specification are based are inadequate for the development of applications that handle data that have multiple languages or cultural environments. This issue is discussed in Section 9.4 on page 65.

(C-06) The requirements for implementations of SQL and CLI to document the character sets and encodings (the codesets) that they can accept should be stated clearly. This issue is discussed in Section 9.5 on page 65 and Section 10.2.2 on page 72.

(C-07) There are issues that require clarification relating to the use of codesets that permit a single graphic symbol to be represented by a combination of codeset elements. This issue is discussed in Section 9.6 on page 67.

(C-08) There are issues relating to the use of UNICODE and ISO 10646 in conjunction with the C programming language that require clarification. This issue is discussed in Section 9.7 on page 67.

11.2 Recommendations

The above conclusions lead to the following recommendations.

- X/Open should draw the attention of standards bodies to the internationalisation problems that result from the fact that the character set in which dynamically executed SQL statements are written is not locale-dependent (see conclusion (C-02)), and should consider requiring one or both of the following solutions in X/Open-compliant systems.
  - — X/Open could state in its **SQL** specification and **CLI** specification that this character set should be locale-dependent.
  - — X/Open could add the SET NAMES statement to its **SQL** specification and **CLI** specification; it could also require the character set names recognised by this statement to include those of standard locales.
- In any case, it is clearly desirable that SQL applications should be able to refer to the character sets and collation sequences of standard locales, as discussed in Section 9.3 on page 64. Implementors should be encouraged to support the character sets and collation sequences of locales specified by X/Open.
- X/Open should draw the attention of the bodies responsible for the standardisation of RDA to the internationalisation problems that result from the requirements in RDA for certain data to be represented by ASN.1 Visible Strings. It should work with those bodies to define other means of representing such data (see conclusion (C-03)).
- The X/Open Data Management Working Group and the Joint Internationalisation Group should work with each other and with the bodies responsible for the standardisation of SQL to develop alternative methods of returning the information that implementations currently supply in the form of diagnostic message text (see conclusion (C-04)). This could include the use of locale-dependent natural language text.
- X/Open should consider enhancing the **SQL** specification and **CLI** specification to incorporate those features of Intermediate and Full SQL that allow an application to handle data that has multiple language or cultural environments (see conclusion (C-05)). While a requirement to implement Full SQL might be considered to impose too great a burden on implementors at the present time, it might be possible to identify a subset of Full SQL that would provide sufficient internationalisation capabilities, which need not be too expensive to implement.
- The **SQL** specification should require that implementations state clearly in their documentation which codesets are allowed to be used for:
  - — embedded SQL statements
  - — dynamically created executable SQL statements
  - — embedded variables.

  The **CLI** specification should require that implementations state clearly in their documentation which codesets are allowed to be used for C function and COBOL subroutine arguments (see conclusion (C-06)).

- X/Open should work with bodies responsible for the standardisation of codesets and of SQL to clarify the following questions (with particular attention to how UNICODE and ISO 10646 combining characters should be treated):
  - How should collating sequences be defined for encoding schemes that represent single graphic symbols by combinations of codeset elements?
  - How are wildcard characters to work and how are numeric positions within strings to be defined for such encoding schemes?

  (See conclusion (C-07)).
- X/Open should work with bodies responsible for the standardisation of UNICODE, ISO-10646, the C programming language and SQL, to clarify how character strings are to be passed in SQL embedded variables or CLI function arguments between C programs and SQL implementations that use UNICODE or ISO 10646 (see conclusion (C-08)).

X/Open Technical Study

Part 4
X/Open DTP Specifications

X/Open Company Ltd.

Chapter 12

Introduction

The chapters in this part of this technical study consider the impact of internationalisation on the X/Open Distributed Transaction Processing (DTP) specifications. These specifications are at present either snapshots, preliminary specifications or CAE specifications. They consist of the following documents (full details are given in **Referenced Documents** on page xv).

- the **TX** (Transaction Demarcation) specification
- the **XA** specification
- the **XA+** specification.

Structure of This Part

Chapter 13 examines the implications of internationalisation on the DTP specifications listed above.

Chapter 14 presents conclusions and recommendations.

After this, Chapter 15 contains a set of *internationalisation* Change Requests (CRs) for each of the DTP specifications listed above. These are edited versions of standard X/Open Change Requests (CRs), in which the identity of the originator is omitted and the CRs are re-numbered into a sequential scheme, for the purposes of this document.

Note: This version of this technical study does not consider the impact of internationalisation on the following X/Open DTP specifications:

- the **TxRPC** specification
- the **XATMI** specification
- the **CPI-C** specification.

Chapter 13

DTP Specifications

13.1 The TX (Transaction Demarcation) Specification

13.1.1 Overview

The **TX** Specification provides an Application Program (AP) with an Application Programming Interface (API) by which the AP can coordinate global transaction management with a Transaction Manager (TM).

The **TX** specification provides application programmers with an API in the following languages:

- ISO C or Common Usage C
- X/Open COBOL.

These interfaces are functionally identical.

For full details of this interface, see the referenced **TX** (Transaction Demarcation) specification.

13.1.2 Internationalisation Implications

Function Names, Arguments, Characteristics and Return Codes

In both C and Cobol, the TX function names, arguments and return codes have an English language flavour. For example, to instruct the TM about transaction timeout information, the AP uses the *tx_set_transaction_timeout*( ) function in C, or the TXSETTIMEOUT function in COBOL.

Similarly, the timeout value is specified in *timeout* (or **TRANSACTION-TIMEOUT**, and the function ultimately returns [TX_OK] on successful completion.

The TX interface is therefore convenient for English-speaking application programmers but not for those of other nationalities. (The end user is not affected because the TX interface is not directly visible at run time.)

The current version of the X/Open Internationalisation Guide only discusses internationalisation in terms of the end user. It focuses on providing internationalised applications that can modify their behaviour at run time for specific language operation. At the present time, there is no basis for internationalising the names of functions, arguments and return codes such as those provided by the TX interface. Changes to this aspect of the TX interface are therefore beyond the scope of this technical study.

The <tx.h> Header

The **<tx.h>** header defines a public structure called an **XID** to identify a transaction branch. The contents of **XID** are used between all components that take part in a global transaction, within or across TM domains.

The **XID** structure is specified in the **<tx.h>** header as follows:

```
#define XIDDATASIZE 128        /* size in bytes */
struct xid_t {
       long formatID;          /* format identifier */
       long gtrid_length;      /* value not to exceed 64 */
       long bqual_length;      /* value not to exceed 64 */
       char data[XIDDATASIZE]; /* may contain binary data */
       };
typedef struct xid_t XID;
/*
 * A value of -1 in formatID means that the XID is null.
 */
```

Although the field *data* is of type **char** and might, at first sight, be a candidate for conversion to type **wchar_t** to allow for multi-byte character encodings, this is not necessary because its significant length at any moment is defined by *gtrid_length* plus *bqual_length*. It does not rely on being null terminated.

However, Section 4.2 of the **TX** specification states that ‘‘APs may use **XID**s for administrative purposes such as auditing and logging ....’’. It then warns that ‘‘the AP should treat each component of *data* as an arbitrary collection of octets because, for instance, a component may contain binary data as well as printable text’’.

Section 4.2 should contain an additional warning for APs that use **XID**s for administrative purposes such as auditing and logging. Because an **XID** may be encoded using a different, possibly mult-byte, character set to the one specified by the current *locale*, the AP should take care only to record (for these additional purposes) the contents of **XID** as the exact sequence of bits in which it was received. The AP should not rely on being able to interpret these bits as printable characters and should certainly avoid trying to display the value of any **XID** to an end user at run time.

13.2 The XA Specification

13.2.1 Overview

The XA interface is the bidirectional interface between a Transaction Manager (TM) and a Resource Manager (RM). It lets a TM structure the work of RMs into global transactions and coordinate transaction completion or recovery.

The **XA** specification provides an API in the following languages:

- ISO C or Common Usage C.

For full details of this interface, see the referenced **XA** specification.

13.2.2 Internationalisation Implications

Function Names, Arguments, Characteristics and Return Codes

In the same way as already described for the **TX** (Transaction Demarcation) specification (see Section 13.1.2 on page 85 of this technical study), the XA function names, arguments and return codes have an English language flavour, and are therefore convenient for English-speaking software developers but not for those of other nationalities. (The XA interface is not directly visible to either the application programmer or the end user.)

Changes to this aspect of the **XA** specification are therefore beyond the scope of this document.

The <xa.h> Header

The **<xa.h>** header defines a public structure called an **XID** to identify a transaction branch. The contents of XID are used between all components that take part in a global transaction, within or across TM domains.

The **XID** structure is specified in the **<xa.h>** header as follows:

```
#define XIDDATASIZE  128        /* size in bytes */
#define MAXGTRIDSIZE  64        /* maximum size in bytes of gtrid  */
#define MAXBQUALSIZE  64        /* maximum size in bytes of bqual  */
struct xid_t {
       long formatID;          /* format identifier */
       long gtrid_length;      /* value 1-64 */
       long bqual_length;      /* value 1-64 */
       char data[XIDDATASIZE];
       };
typedef struct xid_t XID;
/*
 * A value of -1 in formatID means that the XID is null.
 */
```

Although the field *data* is of type **char** and might, at first sight, be a candidate for conversion to type **wchar_t** to allow for mult-byte character encodings, this is not necessary because its significant length at any moment is defined by *gtrid_length* plus *bqual_length*. It does not rely on being null terminated.

However, unlike the **TX** (Transaction Demarcation) specification, the **XA** specification does not explicitly highlight that the field *data* should be treated as an arbitrary collection of octets that might contain binary data.

Resource Manager Name

The Resource Manager Switch (**xa_switch_t**) includes the field *name[RMNAMESZ]* to contain the name of the resource manager. This field is of type **char** and has a defined length of 32 characters including the null terminator.

The **XA** specification does not say why this field is both fixed length *and* null terminated. The specification also does not define the purpose of this field and the possible ways that it might be used at run time by the different DTP components.

There seem to be two alternative ways of making this field suitable for internationalisation: either

- drop the null terminator, leave the field as a fixed length of 32 characters of type **char**, and specify that all characters must be initialised

  or

- retain the null terminator and define that the field has a maximum length of 32 characters of type **wchar_t**, including the null terminator.

In either case, because the field may be initialised with multi-byte characters encodings, the **XA** specification should warn that the field should be treated as an arbitrary collection of characters that may have been encoded using a different, possibly multi-byte, encoding scheme to the one currently defined for the present *locale*.

In this technical study, the related Change Request (CR XA / I18N-02) specified in Section 15.2 on page 95 takes the second of these two options.

XA Information String

The Transaction Manager (TM) uses the functions *xa_open*( ) and *xa_close*( ) respectively to open and close a Resource Manager (RM). Both functions have an argument, *xa_info*, which points to a null-terminated character string that may contain instance-specific information for the RM. This field requires conversion to type **wchar_t** to allow for information strings that are encoded using multi-byte characters.

13.3 The XA+ Specification

13.3.1 Overview

The XA+ interface provides an interface between a Transaction Manager (TM) and a Communication Resource Manager (CRM) to allow global transaction information to flow across TM domains. It also includes the XA interface described in Section 13.2 on page 87 of this technical study.

The **XA+** specification provides an API in the following languages:

- ISO C or Common Usage C.

For full details of this interface, see the referenced **XA+** specification.

13.3.2 Internationalisation Implications

Function Names, Arguments, Characteristics and Return Codes

In the same way as already described for the **TX** (Transaction Demarcation) specification (see Section 13.1.2 on page 85 of this technical study), the XA+ function names, arguments and return codes have an English language flavour, and are therefore convenient for English-speaking software developers but not for those of other nationalities. (The XA+ interface is not directly visible to either the application programmer or the end user.)

Changes to this aspect of the **XA+** specification are therefore beyond the scope of this document.

The <xa.h> Header

In the same way as already described for the **XA** specification (see Section 13.2.2 on page 87 of this technical study), The **<xa.h>** header in the **XA+** specification defines a public structure called an **XID** to identify a transaction branch.

Like **XA**, the **XA+** specification does not explicitly highlight that the field *data*, which contains the global transaction identifier *gtrid* plus branch qualifier *bqual*, should be treated as an arbitrary collection of octets that might contain binary data.

Resource Manager Name

In the same way as already described for the **XA** specification (see Section 13.2.2 on page 87 of this technical study), the Resource Manager Switch (**xa_switch_t**) includes the field **name[RMNAMESZ]** to contain the name of the resource manager. Like **XA**, the **XA+** specification defines this field as type **char** with a defined length of 32 characters including the null terminator.

The **XA+** specification does not say why this field is both fixed length *and* and null terminated. The specification also does not define the purpose of this field and the possible ways that it might be tested at run time by the different DTP components.

Like **XA**, there are two alternative ways of making this field suitable for internationalisation: either

- drop the null terminator, leave the field as a fixed length of 32 characters of type **char**, and specify that all characters must be initialised

  or

- retain the null terminator and define that the field has a maximum length of 32 characters of type **wchar_t**, including the null terminator.

In either case, because the field may be initialised with multi-byte characters encodings, the **XA+** specification should warn that the field should be treated as an arbitrary collection of characters that may have been encoded using a different, possibly multi-byte, encoding scheme to the one currently defined for the present *locale*.

In this technical study, the related Change Request (CR XA+/I18N-02) specified in Section 15.3 on page 98 takes the second of these two options.

XA Information String

In the same way as already described for the **XA** specification (see Section 13.2.2 on page 87), the functions that the TM uses to open and close an RM, *xa_open*( ) and *xa_close*( ) respectively, have an argument *xa_info* which points to a null-terminated character string that may contain instance-specific information for the RM. This field requires conversion to type **wchar_t** to allow for information strings that are encoded using multi-byte characters.

Blob Data

A Communication Resource Manager (CRM) can use the function *ax_set_branch_info*( ) to save information about a transaction branch. The CRM can later access this information using the function *ax_get_branch_info*( ).

The *blob* argument points to the character string of information to be saved. This argument is of type **char**. Although this field might, at first sight, be a candidate for conversion to type **wchar_t** to allow for mult-byte character encodings, this is not necessary because, in each instance of its use, its length is defined by the argument *blob_len*. Its length does not rely on null termination.

However, the **XA+** specification does not explicitly state that *blob* may contain characters encoded using a different character set to the one being used in the current *locale*, and it does not explicitly highlight that *blob* should be treated as an arbitrary collection of characters that might contain binary data.

The **XA+** specification should also warn implementors that the only valid method of determining the length of *blob* is by reference to *blob_len* and that any reliance on null termination may give unreliable results.

Chapter 14

Conclusions and Recommendations

This chapter summarises the implications of internationalisation requirements on X/Open Distributed Transaction Processing (DTP) specifications and presents conclusions and recommendations.

14.1 Summary

In general, there are few problems in using the X/Open DTP specifications internationally. None of the DTP specifications examined are directly concerned with the control or display of data that is used or manipulated by the end user, and are therefore not directly impacted by the internationalisation requirements of different national languages and cultural conventions.

The changes that are required relate mainly to:

- converting to wide characters (**wchar_t**) those character strings that could be misinterpreted by different DTP components using different, and possibly multi-byte, encoding methods
- inserting clarifications where data stored by one DTP component in an apparently character format should only be treated by connected DTP components as arbitrary collections of binary data.

The changes that this technical study recommends are shown as a set of X/Open Change Requests (CRs) in Chapter 15.

14.2 Function Names, Arguments, Characteristics and Return Codes

As described in Section 13.1.2 on page 85, the function names, arguments and return codes used in the DTP specifications have an English language flavour. For example, to set transaction timeout information in the Transaction Manager (TM), the AP uses the function *tx_set_transaction_timeout*( ).

The DTP interfaces are therefore convenient for English-speaking application programmers and software developers but not for those of other nationalities.

The current version of the X/Open **Internationalisation Guide** only discusses internationalisation in terms of the end user, and changes in this area are therefore beyond the scope of this technical study.

14.3 ISO C and Common Usage C

Chapter 4 of the **XA** and **XA+** specifications states that the **<xa.h>** header file is suitable for both ISO C and Common Usage C implementations.

Chapter 15 of this technical study includes Change Requests (CRs) that specify changing certain fields of type **char** into the wide character type **wchar_t**. Although ISO C supports the **typedef** name **wchar_t**, it not certain that all implementations of Common Usage C support it also. If they do not, then the statement in **XA** and **XA+** that the **<xa.h>** header file is suitable for both ISO C and Common Usage C becomes invalid.

This document does not include CRs that address this point. However, if not all Common Usage C implementations support **wchar_t**, then the **XA** and **XA+** specifications should either remove the statement about Common Usage C, or should qualify the statement by saying: ‘‘... suitable for ISO C and for Common Usage C implementations that support the **typedef** name **wchar_t**’’.

The problem does not affect the **TX** (Transaction Demarcation) specification because no changes to use **wchar_t** are proposed.

Chapter 15

Change Requests for Internationalisation

This chapter contains formal change requests for the X/Open distributed transaction processing specifications.

The X/Open Transaction Processing Working Group is currently evaluating these change requests.

15.1 The TX (Transaction Demarcation) Specification

Document: The **TX (Transaction Demarcation)** Specification
X/Open Preliminary Specification, P209 (October 1992).

Change number: TX/I18N-01

Title: Unique Transaction Identifier (**XID**)

Qualifier: Minor Technical

Rationale: An Application Program (AP) uses *tx_info*( ) to obtain **XID** information to identify in which global transaction it is currently located. After the Transaction Manager (TM) has returned this information, the AP may use it for its own administrative purposes, such as auditing and logging.

Because the *data* component contained within the **XID** may have been encoded using a different, possibly multi-byte, character set to the one specified by the current *locale*, the AP should take care only to record (for such purposes) the contents of *data* as the exact sequence of bits in which it was received. The AP should not rely on being able to interpret these bits as printable characters and should not attempt to display the value of any **XID** to an end user at run time.

Change: In Section 4.2, at the end of the paragraph that currently says:

‘‘An important attribute of the **XID** is global uniqueness, based on the exact order of the bits in the *data* element of the **XID** for the lengths specified. The AP should treat each component of *data* as an arbitrary collection of octets because, for instance, a component may contain binary data as well as printable text.’’

Add the following text:

‘‘Because the *data* component contained within the **XID** may have been encoded using a different, possibly multi-byte, character set to the one specified by the current *locale*, the AP should take care only to record the contents of *data* as the exact sequence of bits in which it was received. The AP should not rely on being able to interpret these bits as printable characters and should not attempt to display the value of any **XID** to an end user at run time.’’

15.2 The XA Specification

Document: The **XA** Specification
X/Open CAE Specification, C193 (October 1991).

Change number: XA/I18N-01

Title: Unique Transaction Identifier (**XID**)

Qualifier: Minor Technical

Rationale: The **<xa.h>** header defines a public structure called an **XID** to identify a transaction branch. The contents of XID are used between all components that take part in a global transaction, within or across TM domains. Unlike the **TX** (Transaction Demarcation) specification, the **XA** specification does not explicitly warn that each component of the *data* portion of the **XID** should be treated as a string of bits rather than as recognisable characters.

Change: In Section 4.2, at the end of the paragraph that currently says:

‘‘Although "**xa.h**" constrains the length and byte alignment of ... that the XID is null.’’

Add the following text:

‘‘An important attribute of the **XID** is global uniqueness, based on the exact order of the bits in the *data* element of the **XID** for the lengths specified. Each component of *data* should be treated as an arbitrary collection of octets because, for instance, a component may contain binary data as well as printable text, and it may have been encoded using a different, and possibly multi-byte, encoding scheme to the one active for the current *locale*.’’

Document: The **XA** Specification
X/Open CAE Specification, C193 (October 1991).

Change number: XA/I18N-02

Title: Internationalised Resource Manager Name

Qualifier: Minor Technical

Rationale: The Resource Manager Switch (**xa_switch_t**) includes the field **name[RMNAMESZ]** to contain the name of the resource manager. This field is of type **char** and is null terminated. This field requires conversion to type **wchar_t** to allow for Resource Managers (RMs) that define their names using multi-byte character encodings.

Change:

i. In the second paragraph of Section 4.3, add the following text:

‘‘The RM name is a field of type **wchar_t** that may contain characters from the character set of any natural language, encoded using any encoding scheme. The TM should not rely on this field being encoded in any particular encoding scheme and should treat its contents only as an arbitrary collection of characters. The TM should not display the field in the form of printable characters to users (who may be working in a different *locale* to the one in which the field was originally encoded).’’

ii. Add the above text also at the end of the second dash item of **Public Information** in section 7.2.

iii. In the switch structure in Section 4.3, change:

```
char name[RMNAMESZ];
                /* name of resource manager */
```

to

```
wchar_t name[RMNAMESZ];
                /* name of resource manager */
```

iv. Make the above change also in the equivalent section of Appendix A.

Document: The **XA** Specification
X / Open CAE Specification, C193 (October 1991).

Change number: XA / I18N-03

Title: XA Information Strings

Qualifier: Minor Technical

Rationale: The functions that the TM uses to open and close an RM, *xa_open*( ) and *xa_close*( ) respectively, have an argument *xa_info* which points to a null-terminated character string that may contain instance-specific information for the RM. This field requires conversion to type **wchar_t** to allow for information strings that are encoded using multi-byte characters.

Change:

i. In Section 4.3, change:

```
int (*xa_open_entry)(char *, int, long);
            /* xa_open function pointer */
int (*xa_close_entry)(char *, int, long);
            /* xa_close function pointer */
```

to:

```
int (*xa_open_entry)(wchar_t *, int, long);
            /* xa_open function pointer */
int (*xa_close_entry)(wchar_t *, int, long);
            /* xa_close function pointer */
```

ii. Make the above change also in the equivalent section of Appendix A.

iii. Make the equivalent change in the **SYNOPSIS** section in the manual page for *xa_close*( ) in Chapter 5.

iv. Make the equivalent change in the **SYNOPSIS** section in the manual page for *xa_open*( ) in Chapter 5.

15.3 The XA+ Specification

Document: The **XA+** Specification
X/Open Snapshot, Version 2, S423 (July 1994).

Change number: XA+/I18N-01

Title: Unique Transaction Identifier (**XID**)

Qualifier: Minor Technical

Rationale: The **<xa.h>** header defines a public structure called an **XID** to identify a transaction branch. The contents of XID are used between all components that take part in a global transaction, within or across TM domains. Unlike the **TX** (Transaction Demarcation) specification, the **XA+** specification does not explicitly warn that each component of the *data* portion of the **XID** should be treated as a string of bits rather than as recognisable characters.

Change: In Section 4.2, at the end of the paragraph that currently says:

‘‘Although **<xa.h>** constrains the length and byte-alignment of ... that the XID is null.’’

Add the following text:

‘‘An important attribute of the **XID** is global uniqueness, based on the exact order of the bits in the *data* element of the **XID** for the lengths specified. Each component of *data* should be treated as an arbitrary collection of octets because, for instance, a component may contain binary data as well as printable text, and it may have been encoded using a different, and possibly multi-byte, encoding scheme to the one active for the current *locale*.’’

Document: The **XA+** Specification
X/Open Snapshot, Version 2, S423 (July 1994).

Change number: XA+/I18N-02

Title: Internationalised Resource Manager Name

Qualifier: Minor Technical

Rationale: The Resource Manager Switch (**xa_switch_t**) includes the field **name[RMNAMESZ]** to contain the name of the resource manager. This field is of type **char** and is null terminated. This field requires conversion to type **wchar_t** to allow for Resource Managers (RMs) that define their names using multi-byte character encodings.

Change:

i. In the second paragraph of Section 4.4, add the following text:

‘‘The RM name is a field of type **wchar_t** that may contain characters from the character set of any natural language, encoded using any encoding scheme. The TM should not rely on this field being encoded in any particular encoding scheme and should treat its contents only as an arbitrary collection of characters. The TM should not display the field in the form of printable characters to users (who may be working in a different *locale* to the one in which the field was originally encoded).’’

ii. Add the above text also at the end of the second dash item of **Public Information** in section 7.2.

iii. In the switch structure in Section 4.4, change:

```
char name[RMNAMESZ];
                /* name of resource manager */
```

to

```
   wchar_t name[RMNAMESZ];
                   /* name of resource manager */
```

iv. Make the above change also in the equivalent section of Appendix A.

Document: The **XA+** Specification
X/Open Snapshot, Version 2, S423 (July 1994).

Change number: XA+/I18N-03

Title: XA Information Strings

Qualifier: Minor Technical

Rationale: The functions that the TM uses to open and close an RM, *xa_open*( ) and *xa_close*( ) respectively, have an argument *xa_info* which points to a null-terminated character string that may contain instance-specific information for the RM. This field requires conversion to type **wchar_t** to allow for information strings that are encoded using multi-byte characters.

Change:

i. In Section 4.4, change:

```
int (*xa_open_entry)(char *, int, long);
            /* xa_open function pointer */
int (*xa_close_entry)(char *, int, long);
            /* xa_close function pointer */
```

to:

```
int (*xa_open_entry)(wchar_t *, int, long);
            /* xa_open function pointer */
int (*xa_close_entry)(wchar_t *, int, long);
            /* xa_close function pointer */
```

ii. Make the above change also in the equivalent section of Appendix A.

iii. Make the equivalent change in the **SYNOPSIS** section in the manual page for *xa_close*( ) in Chapter 5.

iv. Make the equivalent change in the **SYNOPSIS** section in the manual page for *xa_open*( ) in Chapter 5.

Document: The **XA+** Specification
X / Open Snapshot, Version 2, S423 (July 1994).

Change number: XA+ / I18N-04

Title: Transaction Branch Information (*blob* data)

Qualifier: Minor Technical

Rationale: A Communication Resource Manager (CRM) can use the function *ax_set_branch_info*( ) to save information about a transaction branch. The CRM can later access this information using the function *ax_get_branch_info*( ).

The *blob* argument points to the character string of information to be saved. This argument is of type **char**. Its length is defined by the argument *blob_len*. Its does not rely on being null terminated.

However, the **XA+** specification does not explicitly state that *blob* may contain characters encoded using a different character set to the one being used in the current *locale*, and it does not explicitly highlight that *blob* should be treated as an arbitrary collection of characters that might contain binary data.

The **XA+** specification should also warn implementors that the only valid method of determining the length of *blob* is by reference to *blob_len* and that any reliance on null termination may give unreliable results.

Change:

i. In the manual page for *ax_get_branch_info*( ) in Chapter 5, after the paragraph that reads:

‘‘The *blob_len* argument is a pointer to an area in which the transaction manager returns the size of *blob*.’’

Add the following new paragraph:

‘‘The *blob* argument may contain characters encoded using a different character set to the one being used in the current *locale*. It should therefore be treated as an arbitrary collection of characters that might contain binary data. The only valid method of determining the length of *blob* is by reference to *blob_len*. Any reliance on null termination may give unreliable results.’’

ii. In the manual page for *ax_set_branch_info*( ) in Chapter 5, add the same new paragraph as described above.

X/Open Technical Study

Part 5

X/Open Systems Management Specifications

X/Open Company Ltd.

Chapter 16

Introduction

Systems management applications are among those most likely to be affected by internationalisation issues. This is because management of a distributed, multi-national system implies management of resources owned or used by all of the users of the system. These users are likely to work in diverse language and cultural environments. As an added complication, there may be several systems managers, who may themselves work in diverse language and cultural environments. The systems management components must be designed to support this.

The chapters in this part of this technical study consider the impact of internationalisation on the X/Open systems management specifications. These specifications are at present either snapshots, preliminary specifications or CAE specifications.

They consist of the following documents (full details are given in **Referenced Documents** on page xv).

- the **XMP** specification
- the **XMPP** specification
- the **XGDMO** specification
- the Performance Management specification (contained in the **UMA** guide, the **UMA DCI** specification, the **UMA MLI** specification and the **UMA DPD** specification)
- the **XBSA** specification
- the **XSMS** specification.

Structure of This Part

Chapter 17 examines the implications of internationalisation on the systems management specifications listed above.

Chapter 18 presents conclusions and recommendations.

After this, Chapter 19 contains a set of *internationalisation* Change Requests (CRs) for each of the systems management specifications listed above. These are edited versions of standard X/Open Change Requests (CRs), in which the identity of the originator is omitted and the CRs are re-numbered into a sequential scheme, for the purposes of this document.

Chapter 17

Systems Management Specifications

17.1 The XMP Specification

17.1.1 Overview

The **XMP** specification defines an Application Programming Interface (API) to management information services. It can be used in conjunction with the OSI systems management protocol defined in the CMISP standard or with the Internet systems management protocol defined in the SNMP Internet RFC. Its use in conjunction with other protocols is not precluded, but is not specified by the **XMP** specification.

For full details of this interface, see the referenced **XMP** specification.

17.1.2 Internationalisation Implications

Error Messages

The **Error-Message** function returns a text string that describes an error. Its description refers to the ‘‘X/Open Native Language System (NLS)’’, but does not specify how use of the NLS affects the string returned, or constrain the character set that may be used. The length of the string is given explicitly, and the string is null-terminated. Presumably, this means that it is terminated by a single null element of type char; this means that encodings such as ISO 10646 or UNICODE that have character representations containing embedded nulls would cause applications that use the null terminator to behave incorrectly.

Use of Latin Alphabet Strings

The specification of class **Entity-Name** defines attribute **entity** as a printable string. This means that management application names and system names are essentially restricted to using ASCII characters.

Note: The specification of class **Name-String** defines attribute **name-String** as an IA5 string. This does not mean that names are restricted to using the characters of International Alphabet nr. 5 (which is a subset of the ASCII character set), since a name can be of class **DS-DN** or **SNMP-Object-Name**, and these are not restricted to particular character sets.)

String Comparisons

The specification of class **Filter-Item** defines attribute **substrings** whose meaning in an internationalised context is unclear. For example:

- the interpretation of ‘‘Initial Substring’’ and ‘‘Final Substring’’ for languages with mixed directionality requires some thought
- what constitutes a substring is open to question for languages in which a single character can have different representations (for example, <e-acute> / <e>+<acute accent>)?

Similarly, the meanings in an internationalised context of attributes **greater-or-equal** and **less-or-equal** of class **Filter-Item** are unclear (what collating order is assumed?)

17.2 The XMPP Specification

17.2.1 Overview

A protocol profile is the specification of a set of communication protocols, and of options within those protocols, to be used by communicating systems for a particular purpose. The **XMPP** specification defines the protocol profiles to be used for systems management within the X/Open CAE. The profiles include OSI communication protocols, Internet communication protocols, and some proprietary communication protocols. The **XMPP** specification defines the profiles by referring to OSI and Internet protocol and profile specifications.

For full details of this interface, see the referenced **XMPP** specification.

17.2.2 Internationalisation Implications

There are no internationalisation implications for the **XMPP** specification.

17.3 The XGDMO Specification

17.3.1 Overview

Managed objects are abstract representations of parts of computer systems and networks. They are used by systems management applications. Different managed objects are defined for different types of computer systems and networks but, because they have a common format, they can be handled by generic systems management software.

The referenced GDMO standard describes how managed objects should be defined. It specifies templates into which the information for each managed object can be inserted in order to produce a formal definition of it. Although they are formal enough that they can be processed automatically, these descriptions are intended to be read by people.

In the X/Open CAE, management applications can use the API defined in the **XMP** specification to invoke management communications services. In doing so, they will use the API defined in the **XOM** specification to manipulate the complex information structures that are communicated. The implementation of the **XMP** specification can include specific packages that support particular managed objects. Alternatively, it can provide a configuration utility that enables the required packages to be generated at compile time from a formalised description of the managed objects that they must support.

The **XGDMO** specification describes how such a formalised description can be generated mechanically from the templates defined in the GDMO standard. The algorithm described also produces the header files that an application program written in the C programming language will need in order to use the XOM API to manipulate the information structures that represent the managed objects, and generates documentation that describes the packages that it has generated.

For full details of this interface, see the referenced **XGDMO** specification.

17.3.2 Internationalisation Implications

The **XGDMO** specification in effect defines a language translator, and similar issues arise as for a programming language. In particular, there is the question of what character sets can be used in the input to the translator, and of what character sets can appear in its output. This question is not addressed in the **XGDMO** specification.

It is implicitly assumed that the input character set includes the distinct lowercase and uppercase versions of the basic Latin alphabet, the decimal digits, and spacing and punctuation characters. Generated output includes specific characters (as in the "OMP_O-" prefix). OM classes and attributes are identified and alphabetised in ascending order.

Lack of support for characters outside the basic Latin alphabet would mean that the natural names could not be used for managed objects and attributes that have been defined and named in a language and cultural environment other than an English one. This would affect the applications programmer, but should not affect the user of the applications programs.

17.4 The UMA Specifications

17.4.1 Overview

The X/Open Universal Measurement Architecture (UMA) supports the collection, management and reporting of performance data and events. It defines four layers of functionality:

Measurement Application Layer
This consists of the various Measurement Application Programs (MAPs) that provide services for technical support of management goals. Examples of MAPs are performance monitors, capacity planning tools, and tuning advisors.

Data Services Layer
This accepts measurement requests from MAPs and supplies measurement data to the MAPs or to other destinations requested by them. Other destinations can include private files or a facility for access and maintenance of historical data known as UMA Data Storage.

Measurement Control Layer
This schedules and synchronises data collection and supplies the collected data to the Data Services Layer.

Data Capture Layer
This layer is responsible for collecting raw data and supplying it to the Measurement Control Layer.

X/Open has produced the following UMA specifications:

- the **UMA** guide, which provides an overview of the UMA
- the **UMA MLI** specification, which defines:
  - the interface through which the Measurement Application Layer invokes the services of the Data Services Layer
  - the protocol used by implementations of the Data Services Layer in different machines to communicate with each other
- the **UMA DCI** specification, which defines the interface through which the Measurement Control Layer invokes the services of the Data Capture Layer
- the **UMA DPD** specification, which defines the format of data passed across the Measurement Layer Interface, and which may also be used internally within the Data Services Layer.

Note: The interface between the Data Services and Measurement Control layers is not specified.

For full details of these interfaces, see the referenced **UMA** specifications.

17.4.2 Internationalisation Implications

Use of Character Strings in the Data Pool

A number of data pool message fields defined in the **UMA DPD** specification consist of character strings. They include, for example:

- the class name, subclass name and subclass abbreviated name from the Names subclass of the Configuration class
- the command name from the Remote Terminal Monitor Measures subclass of the Response Time class
- the partition name from the Disk Partition Data subclass of the Disk Device Data class.

It is not clear what will happen:

- when messages containing such data are processed by an internationalised application that uses a character set and encoding that is not compatible with those used by the Data Services Layer
- when messages are passed between implementations of the Data Services Layer in different machines that use incompatible character sets and encodings.

The strings are null-terminated, so the use of encodings such as ISO 10646 or UNICODE that include nulls in character encodings is problematic. However, the sizes of the text fields are given in the count that is provided in the Text Descriptor or the size that is provided in the Array Descriptor. The application therefore need not rely on the null terminator when determining the lengths of such fields.

It is probable that the underlying assumption is that all applications will use character sets and encodings that are compatible with basic ASCII, and that the character strings in the messages will only use basic ASCII. If this is so, it is probably true for most systems today, but it does restrict the internationalisation of systems in future.

It is also probable that many of these strings will be the same as the ASCII text strings in labels used in the Data Capture Interface. This facilitates the development of measurement control layer programs, but not in an internationalised context. For example, what would happen if a machine in Japan with a filesystem partition named using Kanji characters was managed by a set of management applications running on a machine in which the only supported character set encoding was ASCII?

Use of Character Strings in the Logical Message Protocol

The Logical Message Protocol is defined in the **UMA MLI** specification for communications between implementations of the Data Services Layer in different machines. A number of the message fields defined for this protocol are text strings (for example, the source and destination fields of the *Create* message). It is not clear what character set encodings are to be used for these fields, and it is not clear what happens when messages are passed between machines that use different character set encodings.

The strings are null-terminated, so the use of encodings such as ISO 10646 or UNICODE that include nulls in character encodings is problematic. The specification should either preclude the use of such encodings or advise applications to rely on the count that is provided in the Text Descriptor or the size that is provided in the Array Descriptor, rather than relying on the null terminator, when determining the lengths of such fields.

Note: The **UMA MLI** specification does not explicitly list these descriptors in the way that the **UMA DPD** specification does, but section 6.3.5 (Variable Length Data) of the **UMA MLI** specification appears to imply that they are present, immediately preceding the fields that they describe.

It is probable that there is an implicit assumption that only basic ASCII encodings will be used. Again, this represents a restriction on the development of internationalised systems.

Textual Descriptions in UDU Control Segments

UMA API messages, called UMA Data Units (UDUs), are defined in the **UMA MLI** specification. They are passed across the Measurement Layer Interface. They can include control segments which, when passed from the Data Services Layer to a MAP, can contain status information.

The bodies of such UDU control segments can contain textual descriptions of problems encountered. These are stated to be ‘‘useful for reporting the condition back to a user’’ but would not be usable by internationalised applications. It would be better to state that applications should use a message cataloguing mechanism to generate status messages.

String Arguments of Measurement Layer Interface Functions

A number of arguments of the functions defined in the **UMA MLI** specification are character strings. It is not clear what character set encodings can be used for these strings.

For an internationalised application, it should be possible to use the character set encoding of the currently established locale.

If this encoding allows a null byte to appear as part of the encoding of a character (as ISO 10646 and UNICODE do), then the use of null terminators for strings becomes problematic. This problem can be avoided by specifying that the arguments are arrays of elements of type **wchar_t**.

String Data Capture Interface Data Types

Metric data objects passed across the Data Capture Interface described in the **UMA DCI** specification can be text strings. It is not clear what character set encodings can be used in such strings. It is also not clear what happens when the strings are passed between different machines that use different character set encodings.

The strings are null-terminated, so the use of encodings such as ISO 10646 or UNICODE that include nulls in character encodings is problematic. The specification should either preclude the use of such encodings or advise applications to rely on the length field that is provided, rather than the null terminator, when determining the lengths of such fields.

Use of ASCII Strings in Data Capture Interface Labels

The **UMA DCI** specification states that it is a goal that any textual information should be capable of supporting an internationalised application. It defines structure type *DCILabel* which contains an ASCII text string and may also contain an internationalised label. The structure of such an internationalised label is unspecified, but it is suggested that it could identify a message catalogue and a message within that catalogue.

Since the structure of an internationalised label is unspecified, it is not possible to write a portable application that uses it. However, the Data Capture Interface is not used by MAPs; it is used by programs of the Measurement Control Layer. In view of the way that strings are used in the Data Pool, in the Measurement Layer Interface, and elsewhere in the Data Capture Interface, it is unlikely that the availability of internationalised labels in the Data Capture Interface can very much affect the degree to which a MAP can be internationalised.

The structure of internationalised labels could be specified, but it does not seem worthwhile to do this except as part of a general solution to the problem of string usage.

17.5 The XBSA Specification

17.5.1 Overview

The **XBSA** Specification defines an API to services that can be used to back-up or to archive data, and to restore the data that has been backed up or archived. It is intended to be used by non-management applications that need back-up or archive services, and by management back-up and archive applications.

For full details of this interface, see the referenced **XBSA** specification.

17.5.2 Internationalisation Implications

Each item of information that is backed-up or archived is created in a particular locale and, more importantly, resides in a file whose name is created in a particular locale. Different users who created the files may have names that assume different locales and may use different locales for their filenames.

An internationalised application uses the locale established by the current user. In the case of the XBSA API, the current user could be a system administrator (in the case of a back-up or archive management application) or an ordinary user (in the case of a non-management application).

The names of users and files are passed over the API in character strings. The strings can be lexicographically compared with each other, and these comparisons can use wildcards.

There is no mention of the locale that is to be used to interpret these strings. (Does ‘‘Ålborg’’ come before ‘‘Amsterdam’’ or after ‘‘Zurich’’?) There is no mention of how wildcards are to be interpreted in locales that include combining characters. (Does ‘‘e*’’ match ‘‘écoutez’’?)

Different names may have been encoded using different character set encodings. For example, one user might name files in EBCDIC while another uses UNICODE. This is more likely to occur in distributed system, in which different computers may have different *native* codesets and character encodings, but may also occur in a single computer that supports multiple locales.

In addition to filenames and information-creator names, there are other attributes that are passed over the API as character strings (for example, domain names and policy set names). Also, rules and schedules are character strings. There is no mention of these strings being in the character set of the current locale.

Although the **XBSA** specification does not say so, the variable length character strings that are passed across the API are presumably null-terminated. This would give problems if the characters are encoded in UNICODE or ISO 10646.

There are also attributes that are date / time specifications in particular formats, for example ddmmyy / hh:mm:ss. This is stated in Section 2.8 (Object Descriptors and the BSA Catalog), but there is no description of the specific attributes concerned.

17.6 The XSMS Specification

17.6.1 Overview

The **XSMS** specification presents a system administration framework that includes both object-oriented programming (based on the **CORBA** architecture and specification) and the X/Open distributed systems management reference model (described in the **XRM** specification).

The framework consists of a set of services. They include both systems management services (such as the *policy management service*) and OMG object management[7] services (such as the *object life cycle service*).

The systems management services are specified using OMG Interface Definition Language (OMG IDL). The OMG object management services that are required for systems management are discussed, but are mostly not specified in detail. Specifications of these services either have already been created by the OMG, or are expected to be created by the OMG in the future.

So that the services can be accessed from shell scripts, a set of type mappings and a set of commands are defined. These effectively specify a shell binding for the OMG IDL.

For full details of this interface, see the referenced **XSMS** specification.

17.6.2 Internationalisation Implications

General

While the **XSMS** specification states the importance of the requirement for internationalisation, it also states that internationalisation requirements for implementations are outside its scope. Internationalisation requirements have not been addressed in detail in the definition of the interfaces.

Names

Types of objects, and instances of those types, have human-readable names. Their purpose is to identify the object types and instances at the user interface. The **XSMS** specification does not define the user interface. The user interface would be implemented by application programs and shell scripts using the IDL interfaces that **XSMS** specification does define. In these interfaces, the names are represented as character strings.

Other entities, including filters and actions, have similar names. These names are also represented by strings.

The **XSMS** specification does not specify how these strings are encoded.

There may be a need in some systems to make the names of object types depend on the current locale. For example, an English administrator might wish to use the term *computer* and a French one might wish to use *ordinateur* to refer to the same type of object. There could also be a need to use locale-dependent names for object instances, just as locale-dependent names may be used for files (see Section 17.5 on page 115).

---

7. The concept of *object* as used by the OMG must be distinguished from the concept of *managed object*. OMG objects may be used as the programming constructs that represent managed objects.

Null-Terminated Strings

The description of the *filter* operation refers to the *value* argument being a null-terminated string. The operation is defined in IDL, and the representation of the string datatype is presumably a feature of the binding of IDL to the programming language (eg. C or C++) that is used. If this binding maps IDL strings to null-terminated strings, then use of UNICODE or ISO 10646 character encodings becomes problematic.

String Arguments to Commands

In the command line interface, strings are typically delimited by white space characters (space, tab, newline etc.) or by quote or double-quote characters. The characters that can be used, and their encodings, depend on the currently established locale.

String Comparisons

The filter operations include string comparisons. The semantics of these operations should be locale-dependent. This is however not discussed in the **XSMS** specification.

Exceptions

The IDL data types used in handling exception conditions are defined in the SysAdminExcept module. They include strings giving resource names, operation names and default messages. The semantics of these strings are not described in the **XSMS** specification. They should be locale-dependent.

Chapter 18

Conclusions and Recommendations

This chapter summarises the implications of internationalisation requirements on X/Open systems management specifications and presents conclusions and recommendations.

18.1 Conclusions

In a multi-national enterprise, the users of a system, and even the managers of a system, may work in a number of different language and cultural environments. Systems management applications must be able to support the management of such systems.

This gives rise to a number of issues in the specifications covered by this part of this technical study. They are discussed in the following subsections.

18.1.1 Character String Identifiers

A number of entities - including managed object classes, managed object instances, systems, files and filters - are identified by character strings. In some cases, those strings are specified to consist of ASCII text. In other cases, the character set and encoding are not specified, but there is no provision for them to depend on the current locale.

This issue affects

- the UMA Data Pool, since character string identifiers can appear in data pool messages
- the UMA logical message protocol, since character string identifiers can appear in fields of messages of this protocol
- the UMA measurement layer interface, since character string identifiers can appear as function arguments in this interface
- the UMA data capture interface, since labels can include text strings
- the Backup/Restore interface, which uses character strings to name files, systems, domains, policy sets and other entities
- the Management Services for an OMG Environment, since character strings are used for names in IDL operations (and in the mappings of those operations to the command line interface) and in exception descriptions.

Use of character strings to identify entities creates problems when the user of the entity and the manager of the entity use different character sets and encodings. It may be difficult for the user, for the manager, or for both of them to work effectively. For example, the manager may be unable to restore a file from archive because his locale does not include some of the characters used in the filename.

There are some cases where it would be wrong to use anything other than a character string. For example, user names are normally stored as character strings, and it is hard to envisage how else they could be stored. Thus, it is necessary to find a solution that enables character strings to be used in some cases, as opposed to a solution that replaces all character strings by some other form of identification.

Where use of the Latin alphabet (as in IA5 strings or printable strings) is an option that is open to the application programmer, but is not essential, the writing of internationalised applications is not precluded. However, writers of applications who assume an English language and cultural environment have a facility at their disposal that is not available to writers of other applications. If any change is made, it should be to add similar facilities for other language and cultural environments, rather than to remove the existing facility.

Where there is no alternative to using the Latin alphabet, the writing of internationalised applications, and of applications that assume language and cultural environments other than English ones, is inhibited. In such cases it is desirable to change the specification, either to require a more general type of string (for example, a graphic string) in place of the IA5 or printable string, or to allow a more general type of string as an alternative. However, where the use of the Latin alphabet string derives from an International Standard, such a change should be made in consultation with the relevant international standards body.

18.1.2 Null-Terminated Strings

The use of null-terminated strings gives problems arising in connection with use of encoding schemes such as ISO 10646 and UNICODE that allow embedded nulls in character encodings.

This issue affects

- the XMP, in which error message strings are null-terminated
- the UMA data pool, in which character string message fields are null-terminated
- the UMA logical message protocol, in which text fields are null-terminated
- the UMA measurement layer interface, in which a number of function arguments are null-terminated strings
- the UMA data capture interface, in which metric data objects can be null-terminated strings
- the XBSA, which allows variable length strings (presumably null-terminated) to be passed across the interface
- the Management Services for an OMG Environment, if the OMG IDL string data type is mapped onto a null-terminated string datatype in a programming language.

18.1.3 Descriptive Text

Descriptive text assumes a particular natural language and locale. It presents problems for system users or managers who do not understand the natural language or who do not use the locale. For example, error messages in Japanese would be incomprehensible to most English or American users, and could in any case probably not be displayed on their terminals.

This affects

- the error messages generated by XMP
- textual descriptions in UMA UDU control segments
- exception descriptions in the Management Services for an OMG Environment specification.

All error messages and other textual descriptions generated by an implementation should be displayed to the user in the language of the current locale. This means that, if they have to be transmitted between different system components before being displayed to the user, they should be held internally in some coded form from which appropriate text for various locales can be generated. The message cataloguing mechanism described in **XPG4** provides a means of achieving this.

18.1.4 String Comparisons

Comparisons between strings should take account of the locale (or locales) in which those strings were defined. This affects:

- the **Filter-Item** class in XMP
- character string names in XBSA
- string comparisons in the Management Services for an OMG Environment specification.

Ideally, comparisons should be based on information, such as collation order, that is contained in the current locale. In practice, there are a number of questions as to how this information should be interpreted, and the necessary information has not all been defined (for example, directionality information for complex text languages is still being studied).

Also, some of these definitions (those in XMP, for example) are based on definitions in International Standards, and X/Open should not adopt a solution unless it is also adopted for the International Standards from which the definitions are derived.

18.1.5 Input and Output Character Sets

The input and output character sets used by language translators may restrict the locales in which the programs that they help to produce can be used. This affects the GDMO-XOM translation specification.

There has been some consideration given by workers in the field of Internationalisation to characterless programming languages, in which the lexical tokens are standardised but their representations in specific character sets are not. At present, however, programming language specifications typically require particular characters to be present in the input character set. They also specify particular keywords, which may convey semantic meaning in particular language and cultural environments (usually English ones). For example, the C programming language requires the input character set to include the basic Latin alphabet, and uses the English word "if" as a keyword.

For the purposes of this technical study, it is assumed that it is reasonable to require the programmer to use a particular character set and particular keywords. (After all, learning a programming language is in many ways like learning a natural language, which has its own alphabet and vocabulary.) However, the programmer must be able to have the facilities needed to produce an application that assumes any particular language and cultural environment or, preferably, is internationalised. This does not mean that the compiler must support all language and cultural environments; it means that the compiler standard does not preclude support for any particular environment.

In the case of the **XGDMO** specification, the position is more complicated, because the output of the translator is not an application program; it is input to other programs, and is used by applications programmers. The principle that applies here is that it should be possible for the programs that use the output of the translator to produce an application that assumes any particular language and cultural environment or, preferably, is internationalised. This is possible with the specification in its current form. However, the recommendations in Section 18.2.5 on page 124 of this technical study are made with the aim of ensuring that implementors make clear their positions with regard to support of character sets.

18.1.6 Use of specific Date/Time Formats

This affects the **XBSA** specification.

Dates and times should be displayed to the user in the format prescribed by that user's current locale. They should be stored internally in a form from which any locale-dependent representation can be generated. However, as it is possible to generate locale dependent representations from any fixed format representation, the **XBSA** specification does not preclude (in this respect) the writing of internationalised applications.

18.2 Recommendations

18.2.1 Character String Identifiers

- In XMP, class **Entity-Name** should be redefined to allow more general strings as representations of entity names.
- Either:

  1. the character strings in the UMA Data Pool, the UMA logical message protocol, the UMA measurement layer interface, the UMA data capture interface, the Backup/Restore interface and the Management Services for an OMG Environment should be tagged with an indication of the locale in which they were created

     or

  2. users working in a multi-locale environment should be advised to use only the characters of the POSIX portable filename character set.

  These alternatives have been presented to the X/Open Systems Management working group. This group did not resolve to undertake the work required for alternative 1. Accordingly, alternative 2, which can easily be implemented, should be adopted.

18.2.2 Null-Terminated Strings

- The description of **Error-Message** in XMP should warn applications programmers not to use the null terminator unless they are sure that the character set encoding does not allow embedded nulls.
- Either:

  1. the use of encodings that can include null bytes should be forbidden for character string message fields in the UMA data pool and the UMA logical message protocol

     or

  2. applications should be advised to rely on the length fields that are provided, rather than the null terminator, when determining the lengths of such fields.

- In the UMA measurement layer interface, the UMA data capture interface and the XBSA, either:

  1. the use of character encodings that can include null bytes should be prohibited

     or

  2. the interface should be modified to use arrays of type **wchar_t** in place of, or as an alternative to, null-terminated arrays of type **char**.

- The possibility that the OMG IDL string data type could be mapped onto a null-terminated string datatype in a programming language, and the internationalisation implications of this, should be discussed with the OMG. The impact on the Management Services for an OMG Environment specification should then be assessed.

18.2.3 Descriptive Text

- The description of **Error-Message** in XMP should state that the string returned is dependent on the current locale.
- The use of textual descriptions in UMA UDU control segments and in OMG Environment Management Services exception descriptions should be deprecated for internationalised applications; they should be encouraged to use message catalogs.

18.2.4 String Comparisons

- No changes should be made to the specifications at this time. X/Open should work with international standards bodies to resolve the issue.

18.2.5 Input and Output Character Sets

- The **XGDMO** specification should require implementations to document the character sets that can be used for input and that will appear in the output.

Chapter 19

Change Requests for Internationalisation

This chapter contains formal change requests for the X/Open systems management specifications.

Note: Some of the change requests contained in this chapter are against pre-publication draft versions of the specifications concerned.

19.1 The XMP Specification

Document: The **XMP** specification
X/Open CAE Specification, P306 (March 1994).

Change number: DL-1

Title: Error Message Text Strings

Qualifier: Minor Technical

Rationale: The **Error-Message** function returns a text string that describes an error. Its description refers to the ‘‘X/Open Native Language System (NLS)’’, but does not specify how use of the NLS affects the string returned, or constrain the character set that may be used. The length of the string is given explicitly, and the string is null-terminated. Presumably, this means that it is terminated by a single null element of type char; this means that encodings such as ISO 10646 or UNICODE that have character representations containing embedded nulls would cause applications that use the null terminator to behave incorrectly.

Change: On the *Error-Message*( ) man page:

i. In the description of the *Length* argument, delete ‘‘This is necessary ... Native Language System).’’

ii. In the description of the *Error-text* result, add the following new paragraph between the two existing paragraphs:

‘‘The language and character set encoding of the message text depend on the currently established locale.’’

iii. In the description of the *Length-return* result, add the following new paragraph:

‘‘Internationalised applications, and other applications that use character set encodings (such as that defined by ISO 10646) that allow embedded nulls, should use the **length_return** value to determine the length of the message string rather than relying on the null terminator.’’

Document:	The **XMP** specification X/Open CAE Specification, P306 (March 1994).
Change number:	DL-2
Title:	Entity name syntax
Qualifier:	Minor Technical
Rationale:	The specification of class **Entity-Name** defines attribute **entity** as a printable string. This means that management application names and system names are essentially restricted to using ASCII characters.
Change:	Change the Value Syntax of attribute entity of class Entity-Name from String(Printable) to String(any).

19.2 The XGDMO Specification

Document: The **XGDMO** specification
X/Open Preliminary Specification, P319 (March 1994).

Change number: DL-3

Title: Character Sets

Qualifier: Minor Technical

Rationale: The **XGDMO** specification in effect defines a language translator, and similar issues arise as for a programming language. In particular, there is the question of what character sets can be used in the input to the translator, and of what character sets can appear in its output. This question is not addressed in the **XGDMO** specification.

It is implicitly assumed that the input character set includes the distinct lowercase and uppercase versions of the basic Latin alphabet, the decimal digits, and spacing and punctuation characters. Generated output includes specific characters (as in the "OMP_O-" prefix). OM classes and attributes are identified and alphabetised in ascending order.

Lack of support for characters outside the basic Latin alphabet would mean that the natural names could not be used for managed objects and attributes that have been defined and named in a language and cultural environment other than an English one. This would affect the applications programmer, but should not affect the user of the applications programs.

Change: Add a new Section 3.6 entitled "Output Character Sets" (and renumber the existing Section 3.6 as 3.7). The new section should contain the following text:

‘‘Implementations may generate outputs using various character set encodings. Each encoding used must satisfy the conditions defined in ISO/IEC 9899 (the ISO C Standard) for source and execution character sets. Each implementation must document the character set encodings that it uses.’’

19.3 The UMA Specifications

Documents: The **UMA DPD** specification
X/Open Preliminary Specification, P435 (Pre-publication Draft).

The **UMA DCI** specification
X/Open Preliminary Specification, P434 (Pre-publication Draft).

The **UMA MLI** specification
X/Open Preliminary Specification, P426 (Pre-publication Draft).

Change number: DL-4

Title: Use of Text Strings in Messages

Qualifier: Minor Technical

Rationale: Character strings are used in fields of data pool messages, in the Logical Message Protocol, in Data Capture Interface data types and in Data Capture Interface labels. It is not clear what will happen if different programs assume different character set encodings when processing these strings.

It might be possible to solve this problem in a way that allows for fully internationalised applications using arbitrary character set encodings. For example, all text strings in messages might be replaced by fields encoded using an extension of the heirarchical scheme used in the Data Capture metric name space. This would require a major revision of the whole UMA. The following change is proposed on the assumption that such a major revision will not be undertaken.

Change:

i. In each of the following sections:

   a. Section 2.4.3 (UMA Type Definitions) of the **UMA DPD** specification, after the definition of *UMATextDescr*,

   b. Chapter 6 (UMA Message and Header Formats) of the **UMA MLI** specification, before the start of section 6.1,

   c. Section 3.3.1.1 (DCILabel) of the **UMA DCI** specification, after the definition of the *DCILabel* structure (and see also below for other changes to this section), and,

   d. Section 3.3.4 (Data Types) of the **UMA DCI** specification, after the definition of *DCITextDescr*,

   insert the following new paragraph:

   ‘‘Note that all programs that use this data must assume the same character set and encoding scheme. Where different character sets are in use (for example, in distributed multinational systems), it is recommended that text strings use only the characters of the POSIX portable file name character set. It is recognised that this pragmatic recommendation places some limitations on the degree to which applications can be internationalised.’’

ii. In Section 1.3.1 (Goals) of the **UMA DCI** specification, delete the "internationalisation" list item.

iii. Delete Section 1.3.10 (Internationalisation) of the **UMA DCI** specification.

iv. In Section 3.3.1.1 (DCILabel) of the **UMA DCI** specification, change ‘‘The label attributes structure ... The DCILabel structure is:’’ to the following text:

‘‘The label attributes structure contains a variable length text field (an array of type *char*, null-terminated) and a field that gives the size of the text field (the number of elements in the array, including the null terminator). The field must be padded out to a four byte boundary. The DCILabel structure is:’’

v. In the definition of *DCILabel* in Section 3.3.1.1 (DCILabel) of the **UMA DCI** specification:

   a. Delete the line defining field ‘‘ascii’’

   b. On the next line (defining field "i18n"), change ‘‘i18n’’ to ‘‘desc’’ and delete ‘‘ for I18N’’

   c. On the following line (defining field ‘‘data’’), delete ‘‘ for ascii and i18n’’

vi. Delete the paragraph following the definition of *DCILabel* in Section 3.3.1.1 (DCILabel) of the **UMA DCI** specification (‘‘The internationalised label ... four byte boundary.’’)

Documents: The **UMA DPD** specification
X/Open Preliminary Specification, P435 (Pre-publication Draft).

The **UMA DCI** specification
X/Open Preliminary Specification, P434 (Pre-publication Draft).

The **UMA MLI** specification
X/Open Preliminary Specification, P426 (Pre-publication Draft).

Change number: DL-5

Title: Null-Terminated Message Strings

Qualifier: Minor Technical

Rationale: When null-terminated strings are used, the use of encodings such as ISO 10646 or UNICODE that include nulls in character encodings is problematic. The specification should either preclude the use of such encodings or advise applications to rely on the length field that is provided, rather than the null terminator, when determining the lengths of such fields.

Change: Two (mutually exclusive) alternatives are proposed. They affect the following text:

a. Section 2.4.3 (UMA Type Definitions) of the **UMA DPD** specification, after the definition of *UMATextDescr*

b. Chapter 6 (UMA Message and Header Formats) of the **UMA MLI** specification, before the start of Section 6.1

c. Section 3.3.1.1 (DCILabel) of the **UMA DCI** specification, after the definition of the *DCILabel* structure (and see also below for other changes to this section)

d. Section 3.3.4 (Data Types) of the **UMA DCI** specification, after the definition of *DCITextDescr*.

The alternatives are:

i. When making the change requested by CR DL-4, use the following text in place of that given in CR DL-4:

‘‘Note that all programs that use this data must assume the same character set and encoding scheme. This scheme must not allow embedded nulls in character encodings. Where different character sets are in use (for example, in distributed multinational systems), it is recommended that text strings use only the characters of the POSIX portable file name character set. It is recognised that these pragmatic recommendations place some limitations on the degree to which applications can be internationalised.’’

ii. Insert the following text, as a separate paragraph, before the text requested by CR DL-4:

‘‘Some character set encodings allow embedded nulls. Unless an application can assume that it is not dealing with such an encoding, it should not rely on the null terminator to determine the length of the string.’’

Document:	The **UMA MLI** specification X/Open Preliminary Specification, P426 (Pre-publication Draft).
Change number:	DL-6
Title:	MLI Function Arguments
Qualifier:	Minor Technical
Rationale:	A number of arguments of the functions defined in the **UMA MLI** specification are character strings. It is not clear what character set encodings can be used for these strings.
Change:	In Section 5.2 (MLI Call Parameters), insert the following paragraph before the description of the first parameter (*attrpairs*). ‘‘For string parameters, the character set and encoding used will be that of the currently established locale. Note that all programs that use this data must assume the same character set and encoding scheme. Where different character sets are in use (for example, in distributed multinational systems), it is recommended that text strings use only the characters of the POSIX portable file name character set. It is recognised that this pragmatic recommendation places some limitations on the degree to which applications can be internationalised.’’

Document: The **UMA MLI** specification
X/Open Preliminary Specification, P426 (Pre-publication Draft).

Change number: DL-7

Title: Null-Terminated Function Arguments

Qualifier: Minor Technical

Rationale: When null-terminated strings are used, the use of encodings such as ISO 10646 or UNICODE that include nulls in character encodings is problematic.

Change: Three mutually exclusive possible changes are proposed:

i. When making the change to Section 5.2 (MLI Call Parameters) requested by CR DL-6, use the following text in place of that given in CR DL-6:

  ‘‘For string parameters, the character set and encoding used will be that of the currently established locale. The encoding scheme must not allow embedded nulls in character encodings. Note that all programs that use this data must assume the same character set and encoding scheme. Where different character sets are in use (for example, in distributed multinational systems), it is recommended that text strings use only the characters of the POSIX portable file name character set. It is recognised that these pragmatic recommendations place some limitations on the degree to which applications can be internationalised.’’

ii. Change all function arguments that are of type **char*** to be of type **wchar_t***.

iii. For each function that has arguments that are of type **char***, keep that function unchanged, but introduce an additional function that performs the same tasks but with the char* arguments replaced by arguments of type **wchar_t***.

Document: The **UMA MLI** specification X/Open Preliminary Specification, P426 (Pre-publication Draft).

Change number: DL-8

Title: UDU Error Descriptions

Qualifier: Minor Technical

Rationale: The bodies of UDU control segments can contain textual descriptions of problems encountered. These are not usable by internationalised applications.

Change: In Section 6.2 (UDU Control Segments) replace the text:

‘‘In addition, ... user’s terminal’’

by

‘‘The application should use the catalogue mechanism to generate a textual description of the problem in the language of the currently established locale’’.

19.4 The XBSA Specification

Document: The **XBSA** specification
X/Open Preliminary Specification, P424 (Pre-publication Draft).

Change number: DL-9

Title: Use of Character Strings

Qualifier: Minor Technical

Rationale: Names of users, files and other entities (such as domains and policy sets) are passed across the API in character strings. Different users and applications will create these names in different locales. Difficulties will arise if the locale assumed by the implementation to process a name is not the same as the locale in which the name was created. This could happen if, for example, a system administration facility running in one locale is used to archive or restore files that have been created (and named) in other locales.

Change: Add a new Section 2.7 entitled ‘‘Internationalisation’’ (and renumber the remaining subsections of Section 2). The new section should contain the following text:

‘‘Names of users, files and other entities (such as domains and policy sets) are passed across the API in character strings. Different users and applications will create these names in different locales. Difficulties will arise if the locale assumed by the implementation to process a name is not the same as the locale in which the name was created. This could happen if, for example, a system administration facility running in one locale is used to backup, archive or restore files that have been created (and named) in other locales.’’

‘‘Implementations of the XBSA will process character strings in a locale-dependent manner. Each function will assume the locale that is established at the time that it is invoked. Applications should ensure that the appropriate locale is established when an XBSA function is called.’’

Document: The **XBSA** specification
X/Open Preliminary Specification, P424 (Pre-publication Draft).

Change number: DL-10

Title: Null-terminated strings

Qualifier: Minor Technical

Rationale: Although the **XBSA** specification does not say so, the variable length character strings that are passed across the API are presumably null-terminated. This will give problems if the characters are encoded in UNICODE or ISO 10646.

Change: Three mutually exclusive possible changes are proposed:

i. When adding the new Section 2.7 requested by Change Request (CR) DL-9, include in it the following text in addition to that given in CR DL-9:

''The encoding scheme of the established locale must not allow embedded nulls in character encodings.''

ii. Change ''char'' to ''wchar_t'' in the definitions of the following types:

Administrator
AdminName
AppUserName
BSAUserName
CGName
CopyGPDest
CopyGPName
Description
DomainName
EventInfo
FilterRuleSet
LGName
MethodName
ObjInfo
ObjectName
PolicySetName
ResourceType

iii. For each function that has arguments that include arrays of characters, keep that function unchanged, but introduce an additional function that performs the same tasks but with the arguments replaced by arguments that have arrays of type **wchar_t** in place of the arrays of type **char**.

19.5 The XSMS Specification

Document: The **XSMS** specification
X/Open Preliminary Specification, P421 (Pre-publication Draft).

Change number: DL-11

Title: Use of Text Strings

Qualifier: Minor Technical

Rationale: Character strings are used in the following:

— Names of objects, object types, filters, actions and other entities

— Arguments to commands in the command line interface

— Exception data types.

Ideally, there should be provision for internationalised applications to use Management Services in an OMG Environment. The possibilities of different users requiring different character sets and encodings, and the possibilities that these encodings could allow embedded nulls, should be taken into account in the architecture of Management Services in an OMG Environment. At present, the architecture does not take these possibilities into account. There are complex issues involved, and the matter requires further study.

Change:

i. Change the title of Chapter 4 to ''Components and Issues Not Addressed''.

ii. Add a new Section 4.3 entitled ''Internationalisation''. The new section should contain the following text:

''There should be provision for internationalised applications to use Management Services in an OMG Environment. Character strings are used in:

— Names of objects, object types, filters, actions and other entities

— Arguments to commands in the command line interface

— Exception data types.''

''The possibilities of different users requiring different character sets and encodings, and the possibility that these encodings could allow embedded nulls, should be taken into account. At present, the architecture does not take these possibilities into account. There are complex issues involved, and the matter requires further study.''

''Meanwhile:

— Implementations should document any restrictions on character sets and encodings that they impose.

— Where different character sets are in use (for example, in distributed multinational systems), it is recommended that text strings use only the characters of the POSIX portable file name character set.

— Applications that may be used internationally should not use the textual descriptions in exception descriptions but should establish and use locale-dependent message catalogues instead.''

''It is recognised that this pragmatic recommendation places limitations on the degree to which applications can be internationalised.''

X/Open Technical Study

Part 6

Glossary and Index

X/Open Company Ltd.

Glossary

ANSI

American National Standards Institute

ANSI C

American National Standards Institute specification of the C programming language

API

In X/Open, an Application Programming Interface. This is a set of services (such as functions in a given programming language) by which the application program communicates with other software components.

ASCII

American Standard Code for Interchange of Information. It is a 7-bit code with no parity recommendation, providing 128 different bit patterns for character representation. The internationally agreed version is called ISO-7 and is specified in ISO/IEC 646.

BMP

Basic Multilingual Plane. ISO IS 10646 is intended to be able to cover the character sets of all languages which 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).

byte

A binary expression forming a basic character combination that usually, but not always, comprises 8 bits.

CAE

X/Open's Common Applications Environment.

CCITT

(Consultative Committee of International Telegraph and Telephone) An international committee whose membership is largely composed of government postal, telephone and telegraph agencies (PTTs). This body is now a division of the **ITU**, and is now called the ITU-T.

Change Request (CR)

In X/Open, a formal presentation of a request to change a document. It has a prescribed set of headings, which identify the document, the originator, the subject, the qualifier (critical/major/minor and technical/editorial), the rationale for the change proposal, and the proposed detailed change(s).

codeset

The bit patterns that constitute the encodings of a character set.

CR

See **Change Request**.

FSS-UTF

UCS Transformation Format. an algorithm which, when applied to an IS 10646 encoding, yields a 1, 2, 3 or 5 octet value which is guaranteed not to contain the ISO 646 encodings of any control character, or of the SPACE or DEL characters.

IEC

International Electrotechnical Commission.

IEEE

In U.S.A., the Institute of Electrical and Electronic Engineers.

interoperability

The ability of software and hardware on multiple machines and from multiple vendors to work together effectively.

internationalization

The provision within a computer program of the capability to make itself adaptable to the requirements of different native languages, cultural environments and coded character sets.

ISO

International Organisation for Standardisation. A standards organisation with the membership composed of the standards organisations from each participating country. ISO working groups generate the OSI Protocol Suite standards.

ISO C

The ISO definition of the C programming language, technically the same as ANSI C.

ITU

International Telecommunications Union. See also **CCITT**.

I18n

Abbreviation for Internationalization, there being 18 letters between the first and last letters in this long word.

locale

The definition of the subset of a user's environment that depends on language and cultural conventions.

octet

8 contiguous bits. The term is used instead of byte to prevent confusion with machines that use non-8-bit bytes.

OSI

Open Systems Interconnection. A set of ISO standards for the interconnection of cooperative (open) computer systems, using the ISO 7-layer reference model.

portability

Machine-independent — applied to software which can be readily ported to different machines.

POSIX

A set of IEEE and ISO standards for a portable operating system, based on UNIX.

presentation

OSI Presentation Layer — the 6th layer in the 7-layer OSI Reference Model. It preserves the meaning of the data transferred between Application Entities, and also provides access to the services of the Session Layer.

protocol

A specification for an agreed procedure to enable exchange of information between cooperating entities, via interfaces which provide the necessary functionality to cover format of messages, data checks, flow control, and error handling. A set of protocols governing the exchange of information between remote systems, and set of interfaces covering the exchange between adjacent protocol levels, are collectively referred to as a protocol hierarchy or protocol stack.

teletype

An electrical typewriter machine equipped with a signal interface through which it provides hard copy output and keyboard input for a computer.

UCS

Universal Multiple-Octet Coded Character Set — defined in ISO 10646.

User Datagram Protocol

The connectionless transport layer protocol of the Internet Protocol Suite.

UTF-8

See **FSS-UTF**.

XPG

X/Open Portability Guide

XSI

X/Open System Interface (to an operating system).

Index