共通デスクトップ環境 プログラマーズ・ガイド (国際化対応編)

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650–960–1300

Part No: 805-5811-10
1998 年 11 月

原典: *Common Desktop Environment: Internationalization Programmer's Guide*

Part No: 805-3916-10

Revision A, September 1998

目次

はじめに

『共通デスクトップ環境プログラマーズ・ガイド (国際化対応編)』は、デスクトップを国際化に対応するための情報を提供し、アプリケーションがさまざまな言語および規則を一貫したユーザ・インタフェースで提供できるようにします。

特に、このマニュアルは次のような情報を提供します。

- 開発者に対して、世界中に配布されるアプリケーションを作成するためのガイドラインおよびヒントを提供します。
- デスクトップ内の異なる階層に渡る国際化対応トピックの全体像を提供します。
- リファレンス・マニュアルと、より詳細なドキュメントに対するポインタを提供します。標準ドキュメントを参照することもあります。

このマニュアルは、すでにあるリファレンス・マニュアルや概念的なドキュメントと重複することなく、特定の国際化対応トピックについてのガイドラインと規則を提供することを目的とします。このマニュアルは、オープン・ソフトウェア環境の特定のコンポーネントや階層ではなく、国際化対応トピックに焦点を絞っています。

対象読者

このマニュアルは、アプリケーション・プログラマおよび開発者とその関連分野の方を対象としてさまざまなレベルの情報を提供します。

このマニュアルの構成

このマニュアルの内容を説明します。

第 1 章

デスクトップの国際化対応とローカライズの概要を、ロケール、フォント、描画、入力、クライアント間通信、ユーザ・ビジュアル・テキストの抽出を含めて説明します。国際化対応基準の重要性についても説明します。

第 2 章

アプリケーションを国際化対応にする際に、一般的に開発者が考慮する必要のある一連のトピックについて説明します。これには、ロケール管理、ローカライズされたリソース、フォント管理、ローカライズされたテキスト・タスク、ローカライズされたテキストのクライアント間通信、国際化対応関数などが含まれます。

第 3 章

分散ネットワークにおいてエンコード文字の処理に関連するトピックについて説明します。国際化対応の分散環境での手引として、開発者にクライアント間の相互運用のための基本原理と例を提供します。

第 4 章

国際化対応アプリケーション、ロケール管理、ローカライズされたテキスト、国際化対応ユーザ・インタフェース言語 (UIL)、ローカライズされたアプリケーションなどのトピックを説明します。

第 5 章

ロケール管理、ローカライズされたテキスト・タスク、フォント・セット・メトリクス、ローカライズされたテキストのクライアント間通信規約、文字セットおよびフォント・セットのエンコーディング、登録情報などのトピックについて説明します。

付録 A

メッセージを記述するための一連のガイドラインです。

関連文書

このマニュアルで紹介するトピックについての追加情報は、次のドキュメントを参照してください。

- ISO C:ISO/IEC 9899: 1990,『*Programming Langages — C*』(technically identical to ANS X3.159-1989, Programming Language C)
- ISO/IEC 9945-1: 1990, (IEEE Standard 1003.1)『*Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]*』
- ISO/IEC DIS 9945-2: 1992, (IEEE Standard 1003.2-Draft) 『*Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities*』
- OSF/Motif 1.2:『*OSF Motif 1.2*』Open Software Foundation, Prentice Hall, 1992, ISBN: 0-13-643115-1
- Scheifler, W. R.,『*X Window System, The Complete Reference to Xlib, Xprotocol, ICCCM, XLFD - X Version 11, Release 5*』 Digital Press, 1992, ISBN: 1-55558-088-2
- X/Open:『*X/Open CAE Specification System Interface Definition*』Issue 4, X/Open Company Ltd., 1992, ISBN: 1-872630-46-4
- X/Open:『*X/Open CAE Specification Commands and Utilities*』Issue 4, X/Open Company Ltd., 1992, ISBN: 1-872630-48-0
- X/Open:『*X/Open CAE Specification System Interface and Headers*』Issue 4, X/Open Company Ltd., 1992, ISBN: 1-872630-47-2
- X/Open:『*X/Open Internationalization Guide*』X/Open Company Ltd., 1992, ISBN: 1-872630-20-0
- ISO/IEC 10646-1: 1993 (E):『*Information Technology - Universal Multi-Octet Coded Character Set (UCS). Part 1: Architecture and Basic Multilingual Plane*』

マニュアルの注文方法

SunDocs™ プログラムでは、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) の 250 冊以上のマニュアルを扱っています。このプログラムを利用して、マニュアルのセットまたは個々のマニュアルをご注文いただけます。

マニュアルのリストと注文方法については、米国 SunExpress™, Inc. のインターネットホームページ `http://www.sun.com/sunexpress` にあるカタログセクションを参照してください。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1 表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、およびディレクトリ名を示します。または、画面上のコンピュータ出力を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `system%`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力とは区別して示します。	`system%` **`su`** `password:`
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。
『 』	参照する書名を示します。	『ユーザーズ・ガイド』を参照してください。
「 」	参照する章や節を示します。また、ボタンやメニューなど、強調する単語を囲む場合にも使用します。	第 1 章「基本スキル」を参照してください。
[ ]	アイコン、ボタン、メニューなどのラベル名を使用します。	[了解] ボタン

注 - \ (バックスラッシュ) は、デバイスによって ¥ (円記号) で表示されるものがあります。

コード例は次のように表示されます。

- C シェルプロンプト

```
system% command [filename]
```

- Bourne シェルおよび Korn シェルのプロンプト

```
system$ command [filename]
```

- スーパーユーザーのプロンプト

```
system# command [filename]
```

[ ]は省略可能な項目を示します。上記の場合、*filename* は省略してもよいことを示します。

ただし AnswerBook2™ では、ユーザーが入力する文字と画面上のコンピュータ出力は区別して表示されません。

一般規則

- このマニュアルでは、英語環境での画面イメージを使っています。このため、実際に日本語環境で表示される画面イメージとこのマニュアルで使っている画面イメージが異なる場合があります。本文中で画面イメージを説明する場合には、日本語のメニュー、ボタン名などの項目名と英語の項目名が適宜、併記されています。

第 1 章

国際化対応について

国際化対応とは、コンピュータ・システムとアプリケーションを世界中のユーザに向けて設計することです。世界のユーザは異なる言語を使用しており、操作するシステムの機能性やユーザ・インタフェースに対する要求事項も異なります。これらの差異にもかかわらず、ユーザは世界中のどの場所でも実行できる企業用のアプリケーションの実現を求めています。そのようなアプリケーションは、国境を越えて相互運用できなければならず、複数のベンダから供給されるさまざまなハードウェア構成で実行でき、かつさまざまな国や地域のユーザの要求を満たすようローカライズされていなければなりません。このオープンな分散コンピューティング環境が、共通オープン・ソフトウェア環境の推進力になっています。このマニュアルで説明されている国際化対応テクノロジは、世界市場に上記のような利点を提供します。

1ページの「国際化対応の概要」

6ページの「ロケール」

8ページの「フォント、フォント・セット、フォント・リスト」

12ページの「テキスト描画」

13ページの「入力メソッド」

20ページの「クライアント間通信規約 (ICCC)」

国際化対応の概要

1 つの共通オープン・システムには、異なる国語をサポートする複数の環境が存在することがあります。そのような 1 つ 1 つの国の環境はロケールと呼ばれ、言語、

文字、フォント、データの入力や書式化の慣習を考慮します。共通デスクトップ環境は、どのアプリケーションでもシステム上にインストールされているすべてのロケールを使用して実行できるように、完全な国際化対応になっています。

ロケールは、プログラムの実行時の動作を、ユーザの地域の言語および慣習に応じて定義します。システム全体を通じて、ロケールは次のことに影響します。

- テキスト・データのエンコーディングおよび処理
- 言語と、リソース・ファイルおよびそのテキスト値のエンコーディングの識別
- テキスト文字列の描画およびレイアウト
- クライアント間テキスト通信に使用されるテキストの交換
- 入力メソッドの選択 (どのコード・セットが生成されるか) およびテキスト・データの処理
- クライアント間テキスト通信のためのエンコードおよびデコード
- ビットマップ・ファイルおよびアイコン・ファイル
- アクションおよびファイル型
- ユーザ・インタフェース定義 (UID) ファイル

国際化対応アプリケーションには、ユーザのロケール、そのロケールを表すのに必要な文字、ユーザが見て対話したいと思う形式 (日付と通貨など) に依存するコードは含まれません。デスクトップはこれを、言語依存情報と文化依存情報をアプリケーションから分離し、アプリケーション外に保存することによって実現しています。

図 1–1 は、国際化対応をシンプルにするためにアプリケーション外に置かれる情報の種類を示します。

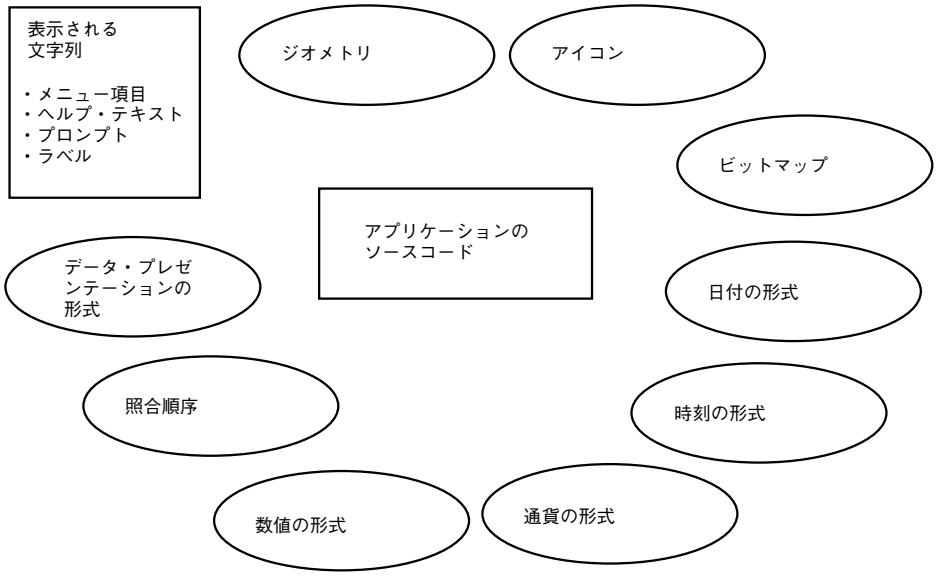

図 1–1　アプリケーションの外の情報

言語依存情報と文化依存情報をアプリケーションのソースコードから分離することにより、異なる国々で販売するためにアプリケーションを再記述したり再コンパイルする必要はありません。その代わり、唯一の要求事項は、地域の言語と慣習に適応するように外部情報がローカライズされなければなりません。

国際化対応アプリケーションは、異なる母国語、地域の慣習、文字列のエンコーディングの要求事項に適合できます。オペレーションを特定の母国語、地域の慣習、文字列のエンコーディングに適合させるプロセスを、ローカリゼーション (L10N) と呼びます。国際化対応の目的は、プログラム・ソースを変更したり再コンパイルしなくてもローカリゼーションが可能になることです。

国際化対応の概要を知るには、『*X/Open CAE Specification System Interface Definition*』Issue 4, X/Open Company Ltd., 1992, ISBN: 1-872630-46-4 を参照してください。

国際化対応の現状

以前は、独自の関数から X/Open の発表する標準関数の新しいセットまで、数多くの種類の国際化対応を業界が提供していました。また、単純な ASCII サポート、ラテン言語／ヨーロッパ言語サポート、アジア言語マルチバイト・サポート、アラビア語／ヘブライ語の両方向サポートなど、レベルもさまざまでした。

X/Open 仕様で定義されたインタフェースは、次のような広範囲の言語および地域をサポートすることができます。

スクリプト	説明
ラテン言語	南北アメリカ、東欧／西欧
ギリシャ語	ギリシャ
トルコ語	トルコ
東アジア	日本語、韓国語、中国語
インド語派	タイ語
両方向	アラビア語とヘブライ語

さらに、共通デスクトップ環境の目的は、これらのテクノロジのローカリゼーション (メッセージおよびマニュアルの翻訳、その他ローカル・ユーザのニーズに合った適合化) を一貫した方法で行うことです。これは、世界中でサポートされているユーザが、ベンダは違っても同じ共通ローカライズ環境を使用できるようにするためです。エンド・ユーザと管理者は、世界中のソフトウェアをサポートするための完全なアプリケーション環境を提供する、一貫性のあるローカライズ機能を期待できます。

国際化対応の標準

多くの企業の努力を通じて、追加の要求事項および言語 (特に東アジアの言語) を盛り込む過程を経て、国際化対応アプリケーション・プログラム・インタフェースの機能性は標準化されてきました。この作業は主に、POSIX (コンピュータ環境用ポータブル・オペレーティング・システム・インタフェース) および X/Open の仕様に集約されてきました。オリジナルの X/Open 仕様は、第 2 版の『*X/Open Portability Guide*』(XPG2) で発表され、それは Hewlett-Packard 社がリリースした母国語サポート・プロダクトに基づいています。最も新しく発表された X/Open 国際化対応の標準は、XPG4 と呼ばれます。

デスクトップ内の各階層が、エンド・ユーザが一貫性のあるローカライズされたインタフェースを確保できるように、国際化対応用に定義された適切な標準インタフェース・セットを使用していることが大切です。ロケールと、ロケール依存関数の共通オープン・セットの定義は、次の仕様書に基づいています。

- 『*X Window System, The Complete Reference to Xlib, Xprotocol, ICCCM, XLFD - X Version, Release 5*』Digital Press, 1992, ISBN 1-55558-088-2.
- 『*ANSI/IEEE Standard Portable Operating System Interface for Computer Environments*』IEEE.
- 『*OSF™ Motif 1.2 Programmer' Reference, Revision 1.2*』Open Software Foundation, Prentice Hall, 1992, ISBN 0-13-643115-1.
- 『*X/Open CAE Specification Commands and Utilities*』Issue 4, X/Open Company Ltd., 1992, ISBN 1-872630-48-0.

この環境内で、ソフトウェア開発者は、移植性が高く、(ベンダが異なっても) 分散システム間で相互運用でき、デスクトップ標準ロケールでサポートされる多国籍ユーザの多様な言語および文化の要求事項を満たす、世界共通のアプリケーションの開発を期待できます。

共通国際化対応システム

図 1–2 は、国際化対応が特定のシングルホスト・システムに広がる様子を示しています。ゴールは、下位のシステムによってサポートされているロケールのセットに対して、世界中で出荷されるよう、アプリケーション (クライアント) が構築されることです。標準インタフェースを使用すると、世界市場への投入しやすさが改善され、アプリケーション開発者が必要とするローカリゼーション作業の量を最小限に減らすことができます。さらに、それぞれの国において、デスクトップの原則を守るシステムで一貫性のあるローカリゼーションを保証できます。

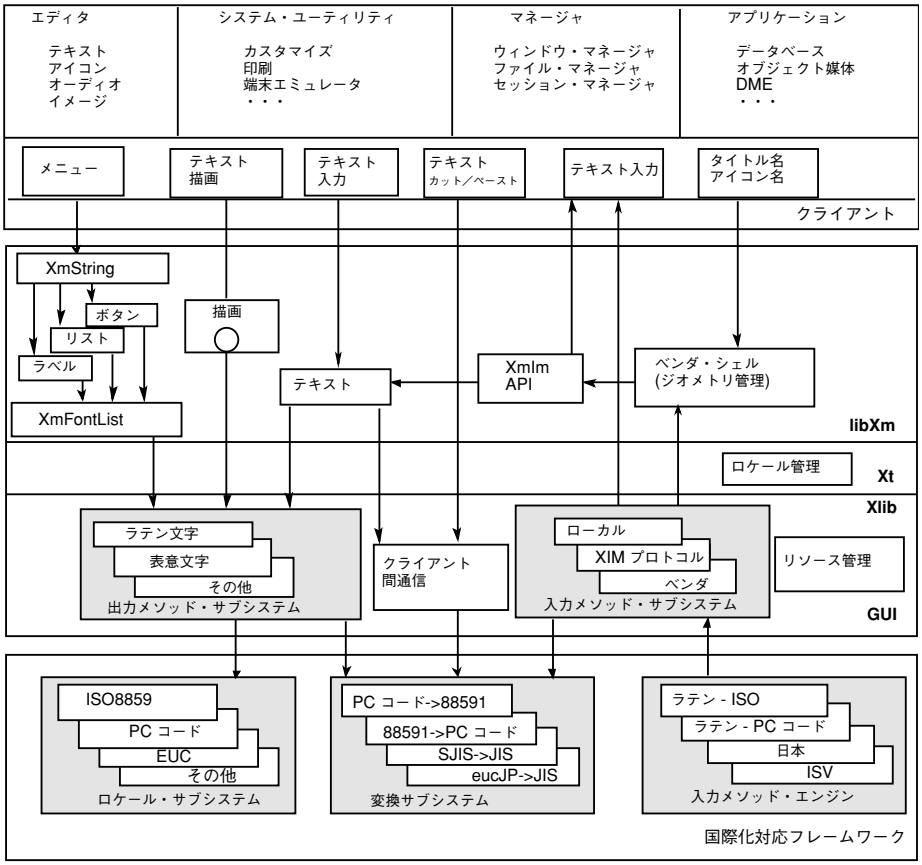

図 1–2　共通国際化対応システム

ロケール

ほとんどの単一画面のクライアントは、実行時に環境変数 (通常は `$LANG` または `xnlLanguage` リソース) の設定で決定される単一のロケールで動作します。環境を制御するには `LC_ALL`、`LC_CTYPE`、`LANG` などのロケール環境変数を使用できます。詳細は、第 5 章の 109ページの「Xt ロケール管理」を参照してください。

ロケールの `LC_CTYPE` カテゴリは、環境によって、実行時に使用されるロケール固有の機能を識別するのに使用されます。ツールキットにより読み込まれるフォントおよび入力メソッドは、`LC_CTYPE` カテゴリで決定されます。

国際化対応されたプログラムは、ユーザの希望するロケールを設定するために XtSetLanguageProc()関数 (デフォルトでは setlocale()) を呼び出すようになっています。ロケールを設定するために setlocale()関数を呼び出すライブラリはないので、特定のロケールまたは実行時に読み込まれた値で XtSetLanguageProc()を呼び出すのはアプリケーションの責任です。アプリケーションが国際化対応なのにXtSetLanguageProc()を使用しない場合は、次の優先順位のソースのいずれかから、setlocale()関数に渡すロケール名を獲得してください。

- コマンド行オプション
- リソース
- 空の文字列 (“”)

空の文字列を指定すると、setlocale() 関数がロケールの設定を決定するのに環境変数 $LC_* と $LANG を使用します。特に、setlocale (LC_ALL,“”) は、表 1–1 に示す順にさまざまなロケール・カテゴリの環境変数のためにロケールがチェックされ選択されます。

表 1–1 ロケール・カテゴリ

カテゴリ	第 1 環境変数	第 2 環境変数	第 3 環境変数
LC_CTYPE:	LC_ALL	LC_TYPE	LANG
LC_COLLATE:	LC_ALL	LC_COLLATE	LANG
LC_TIME:	LC_ALL	LC_TIME	LANG
LC_NUMERIC:	LC_ALL	LC_NUMERIC	LANG
LC_MONETARY:	LC_ALL	LC_MONETARY	LANG
LC_MESSAGES:	LC_ALL	LC_MESSAGES	LANG

ツールキットはすでに標準のコマンド行オプション (–xnllanguage) およびリソース (xnlLanguage) を定義しています。また、リソースの値は RESOURCE_MANAGER サーバでも設定できます。その場合、RESOURCE_MANAGER サーバに接続するすべてのクライアントに影響する可能性があります。

フォント、フォント・セット、フォント・リスト

すべての X クライアントはテキストを描画するのにフォントを使用します。テキスト描画に使用する基本的なオブジェクトは `XFontStruct()` です。`XFontStruct()` は、描画するイメージを含むフォントを識別します。

すでにデスクトップは、Xlib で定義される `XFontStruct()` データ構造体としてフォントをサポートしています。しかし、フォント内の文字のエンコーディングは国際化対応アプリケーションに認識されていなければなりません。この情報を知るために、プログラムはサーバのすべてのフォントが X 論理フォント (XLFD) 名で識別できることを期待します。XLFD 名により、ユーザは基本特性と charset (フォント・グリフのエンコーディング) の両方を記述できます。charset という用語は、フォント内のグリフのエンコーディングを表すのに使用されます。一方、コード・セットという用語はロケール内の文字のエンコーディングを意味します。指定されたフォントの charset は、XLFD 名の CharSetRegistry フィールドと CharSetEncoding フィールドで決定されます。テキストと記号は、フォント内のコードによって定義されたとおりに描画されます。

フォント・セット (例: Xlib によって定義されるデータ構造体 `XFontSet()`) は、指定のロケール用に定義されたすべての文字を描画可能にする 1 つ以上のフォントの集合です。ロケールによってはグリフの索引とコード・セットのエンコーディングが一致しない場合がありますが、国際化されたアプリケーションは、このような場合でもテキストを描画できなければなりません。さらに、エンコーディングがロケールのコード・セットと異なるフォントを 1 つ以上使用しているロケールのすべての文字を描画するには、複数のフォントが必要になることがあります。コード・セットと charset は両方ともロケールごとに異なるため、フォント・セットという概念を `XFontSet()` として実現しています。

フォントが XLFD 名によって識別される一方、フォント・セットは XLFD 名のリストによって識別されます。基本特性のみが重要である点を除き、リストは 1 つ以上の XLFD 名から成ります。必要とされるフォントのエンコーディングはロケールから決定されます。XLFD ベース名リストに指定されている charset はすべて無視されるため、ユーザが考慮する必要があるのはポイント・サイズ、スタイル、ウェートなどのベース特性を指定することだけです。フォント・セットはロケールによって変わり、ロケールのコード・セットでエンコードされたテキストを描画するのに使用されます。国際化対応アプリケーションは、テキスト・データを描画するのにフォント構造体の代わりにフォント・セットを使用すべきです。

フォント・リストは、1 つ以上のフォント・リスト・エントリの集合である libXm Toolkit オブジェクトです。フォント・セットはフォント・リストで指定できます。各フォント・リスト・エントリは、フォントかフォント・セットのいずれかを指定し、名前がタグ付けされます。フォント・リスト・エントリにタグがない場合は、デフォルト・タグ (XmFONTLIST_DEFAULT_TAG) が使用されます。フォント・リストは、libXm Toolkit ライブラリにある XmString() 関数と共に使用できます。フォント・リストにより、1 つ以上のセグメント (各セグメントはタグで識別される) から成るコンパウンド・ストリングの描画が可能になります。これにより、異なる基本特性を持つ文字列の描画が可能になります (たとえば、1 回のオペレーションでボールドとイタリックの文字列を描画できます)。libXm ライブラリの XmText() など一部の XmString() ベースではないウィジェットは、フォント・リストでフォント・リスト・エントリを 1 つしか使用しません。Motif フォント・リストは、フォント・リスト内のフォント・セットを識別するために接尾辞 : (コロン) を最後につけます。

通常、ユーザは (フォントかフォント・セットのいずれかが含まれている) フォント・リストか、またはフォント・セットを指定するよう要求されます。国際化対応環境では、ユーザはコード・セットに依存しないフォントを指定できなければなりません。というのは、その指定は、フォントの文字セット (charset) よりも、異なるコード・セットを持つさまざまなロケールで使用されるからです。したがって、すべてのフォント・リストにはフォント・セットを指定するようにしてください。

フォント指定

フォント指定は、X 論理フォント (XLFD) 名か、XLFD 名の別名のいずれかになります。たとえば、次の例はどちらも 14 ポイント・フォントの有効なフォント指定です。

```
-dt-application-medium-r-normal-serif-*-*-*-*-p-*-iso8859-1
```

または

```
-*-r-*-14-*iso8859-1
```

フォント・セット指定

フォント・セット指定は、名前 (XLFD 名かその別名) のリストであり、ベース名リストと呼ばれることもあります。すべての名前はカンマで区切られ、カンマの前後にある空白スペースはすべて無視されます。XLFD 名を短縮するためにパターン照合 (ワイルドカード) 文字を指定できます。

フォント・セット指定は、実行中のロケールによって決定されることに注意してください。たとえば、日本語ロケール ja_JP は、日本語のすべての文字を表示するのに必要な 3 つのフォント (文字セット) を定義します。次の例では必要なゴシック・フォントのセットが識別されます。

- 完全 XLFD 名リストの例

```
-dt-mincho-medium-r-normal--14-*-*-m-*-jisx0201.1976-0,
-dt-mincho-medium-r-normal--28-*-*-*-m-*-jisx0208.1983-0:
```

- 単一 XLFD パターン名の例

```
-dt-*-medium-*-24-*-m-*:
```

上記の 2 例は、ベース名リストに一致するフォントが存在する限り日本語ロケールで使用できます。

フォント・リスト指定

フォント・リスト指定は 1 つ以上のエントリから成り、各エントリはフォント指定かフォント・セット指定のいずれかになります。

各エントリには、コンパウンド・ストリングを描画するときに使用される名前がタグ付けされます。タグはアプリケーションで定義され、通常はフォントの種類が予想できるような名前です (`bold()`、`italic()`、`bigbold()` など)。ヌルのタグはデフォルト・エントリを表すのに使用され、`XmString()` 関数で使用される `XmFONTLIST_DEFAULT_TAG` 識別子に関連付けられます。

フォント・タグは、= (等号記号) が接頭部に付くときに識別されます。たとえば、`=bigbold()` はサーバで定義された最初のフォントに一致します。= が指定されていてもその後に名前がない場合は、その指定は「デフォルト・フォント・リスト・エントリ」と見なされます。

フォント・セット・タグは、: (コロン) が接頭部に付くときに識別されます。たとえば、`:bigbold()` はロケールの条件を満たす、サーバの最初のフォント・セットに一致します。: が指定されていても名前が指定されていない場合は、その指定はデフォルト・フォント・リスト・エントリと見なされます。フォント・リスト・エントリ指定内では、ベース名リストは , (カンマ) ではなく ; (セミコロン) で区切られます。

フォント・リスト指定の例

ラテン 1 ロケール用には、次のように入力します。

```
-*-r-*-14-*: ,      # default font list entry
-*-b-*-18-*:bigbold # Large Bold fonts
```

ベース・フォント名リスト指定

ベース・フォント名リストは、ロケールによって定義されたフォント・セットに関連付けられたベース・フォント名のリストです。ベース・フォント名はカンマで区切られたリストであり、ポータブル文字セットからの文字だと想定されます。そうでない場合の結果は不定です。セパレータのカンマに隣接する空白スペースは無視されます。

XLFD フォント名の使用により、国際化対応のアプリケーションが、単一のロケールに依存しないベース・フォント名からさまざまなロケールに必要なフォントを得ることができます。単一のベース・フォント名は、該当ロケールに必要なさまざまな charset でメンバがエンコードされたフォントのファミリを指します。

XLFD ベース・フォント名は、ロケールに必要なフォントの charset を明示的に指定できます。このことにより、ユーザはロケールに必要な charset で使用するフォントを厳密に指定することができるので、フォント選択を完全に制御できます。

ベース・フォント名が XLFD 名でない場合は、フォントのフォント属性から XLFD 名を獲得しようとします。

次のアルゴリズムは、フォント・セットでテキストを表示するのに使うフォントを選択するために使用されます。

ロケールに必要な各 charset ごとに、ベース・フォント名リストはサーバに存在するフォント・セットを指定する以下の場合の最初にあてはまるものが検索されます。

- 必要な charset または必要な charset のスーパーセットを、CharSetRegistry と CharSetEncoding フィールドに持つ最初の XLFD 準拠ベース・フォント名
- 必要な charset をサポートするために再マップできる 1 つ以上の charset を指定する、1 つ以上の XLFD 準拠ベース・フォント名の最初のセット。Xlib 処理系は、必要な charset から 1 つ以上のその他の charset へのさまざまなマッピングを認識できます。また、それらの charset のフォントを使用できます。たとえば、JIS ローマンは、ASCII の ~ (チルダ) と \ (バックスラッシュ) の代わりに ¥ (円記号) と ‾ (オーバーバー) を使用したものです。Xlib は、JIS ローマン・フォントが使用できない場合、この文字セットをサポートするために ISO8859-1 フォントを読み込めます。

- (XLFD フォント名の CharSetRegistry と CharSetEncoding フィールドの代わりに) 必要な charset と組み合わされた最初の XLFD 準拠フォント名、または XLFD フォント名を獲得できる最初の XLFD ではないフォント名。最初の手続きで、処理系は必要な charset のスーパーセットである charset を使用できます。
- ロケールに依存した何らかの方法で charset を連想させるテキストをサポートする 1 つ以上のフォントにマップできる、最初のフォント名

たとえば、ロケールには次の charset が必要だと想定します。

- ISO8859-1
- JISX0208.1983
- JISX0201.1976
- GB2312-1980.0

次の例のように、charset を明示的に指定したベース・フォント名リストを提供し、特定のフォントが存在する場合には確実に使用するようにできます。

```
"-dt-mincho-Medium-R-Normal-*-*-*-*-*-M-*-JISX0208.1983-0,\
-dt-mincho-Medium-R-Normal-*-*-*-*-*-M-*-JISX0201.jisx0201\.1976-1,\
-dt-song-Medium-R-Normal-*-*-*-*-*-M-*-GB2312-1980.0,\
-*-default-Bold-R-Normal-*-*-*-*-M-*-ISO8859-1"
```

次の例のように、charset を省いたベース・フォント名リストを提供すると、必要な各コード・セット用のフォントを選択できます。

```
"-dt-Fixed-Medium-R-Normal-*-*-*-*-*-M-*,\
-dt-Fixed-Medium-R-Normal-*-*-*-*-*-M-*,\
-dt-Fixed-Medium-R-Normal-*-*-*-*-*-M-*,\
-*-Courier-Bold-R-Normal-*-*-*-*-M-*"
```

代わりの方法として、次の例のように単一ベース・フォント名を提供すると、ある最小の XLFD 属性要求事項を満たす使用可能なすべてのフォントから選択できます。

```
"-*-*-*-R-Normal--*-*-*-*-*-M-*"
```

テキスト描画

デスクトップは、シンプルなテキスト、コンパウンド・ストリング、数種類のウィジェットを含むローカライズされたテキストを受け渡すさまざまな関数を提供します。これらの中には Xlib ライブラリと Motif ライブラリの関数も含まれます。

入力メソッド

共通デスクトップ環境は、Xm Toolkit を使用する国際化対応したアプリケーションに対して、ローカライズされた入力を行う機能を提供します。特に、XmText[Field]() ウィジェットが、各ロケールで提供される入力メソッドとインタフェースすることが可能になります。さらに、dtterm() クライアントも、入力メソッドを使用することが可能になります。

デフォルトでは、libXm Toolkit を使用するそれぞれの国際化対応クライアントは、ユーザの指定したロケールに関連付けられた入力メソッドを使用します。ユーザが代替入力メソッドを自由に指定できるように、XmNinputMethod() リソースがロケール名のモディファイアとして提供されます。

入力メソッドのユーザ・インタフェースは複数の要素から構成されています。それらの領域の必要性は、使用されている入力メソッドによります。通常それらの要素は、複雑な入力処理とダイアログを要求する入力メソッドの場合に必要となります。

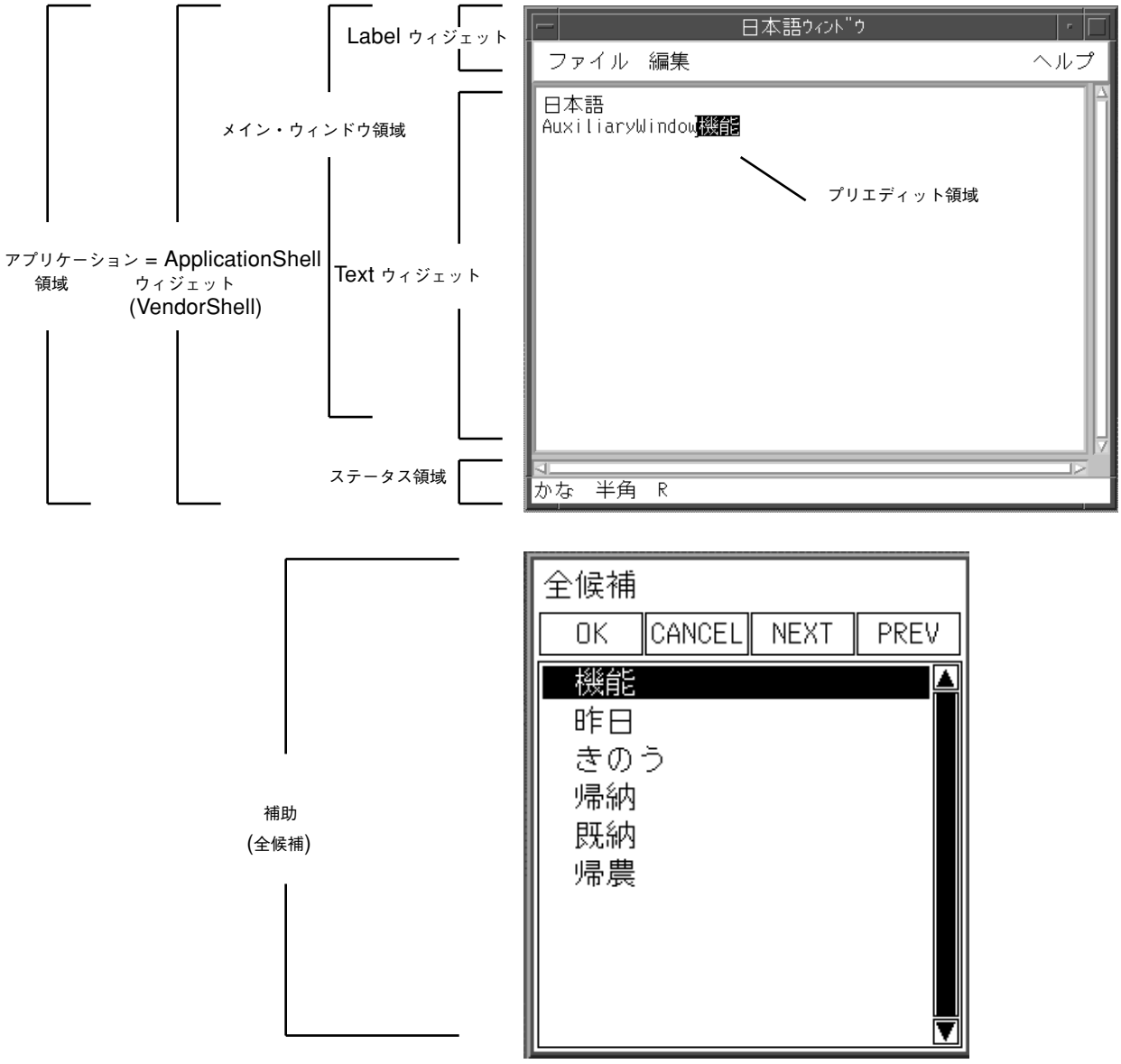

図 1–3 VendorShell ウィジェットと補助の例 (日本語)

プリエディット領域

プリエディット領域は、あらかじめ編集される (プリエディット) 文字列を表示するのに使用されます。入力メソッドは、次の 4 つのプリエディット・モードをサポートしています。OffTheSpot、OverTheSpot (デフォルト)、Root、および None です。

注 - 確定した文字列は再変換できません。文字列の状態は、プリエディット領域から、ユーザが文字を入力している位置へ移動します。

OffTheSpot

入力メソッドを使用する OffTheSpot モードのプリエディットでは、プリエディットの位置は図 1–4 のようにメイン・ウィンドウ領域のすぐ下かつステータス領域の右側に固定されています。日本語の入力メソッドを例示します。

図 1–4　VendorShell ウィジェットでの OffTheSpot プリエディットの例 (日本語)

システム環境では、入力メソッドを使用してプリエディットすると、編集されているプリエディット文字列が、入力メソッドによって何らかの形で強調表示されます。

OffTheSpot モードを使用するには、`VendorShell()`ウィジェットの`XmNpreeditType()` リソースを、`XtSetValues()` 関数かリソース・ファイルのどちらかで設定します。`XmNpreeditType()` リソース

は、TopLevelShell()、ApplicationShell()、DialogShell() ウィジェットのリソースとしても設定できます。この 3 つのウィジェットは、VendorShell() ウィジェット・クラスのサブクラスです。

OverTheSpot (デフォルト)

OverTheSpot モードでは、プリエディット領域の位置はユーザが文字を入力しようとする場所 (たとえば現在のフォーカスを持つ Text ウィジェットの挿入カーソルの位置) に設定されています。プリエディット領域の文字は、カーソル位置にオーバレイ・ウィンドウとして表示され、入力メソッドによっては強調表示されます。

OverTheSpot モードでは 1 つのプリエディット領域が複数の行から成る場合があります。プリエディット領域は常にメイン・ウィンドウ領域の中にあり、どの方式でもはみ出すことはありません。

プリエディット中の文字列が Text ウィジェットのテキストの一部であるかのように表示されていても、プリエディットが終了するまでは、クライアントに渡されて下位の編集画面に表示されることはないので注意してください。図 1–5 を参照してください。

OverTheSpot モードを明示的に使用するには、VendorShell() ウィジェットの XmNpreeditType() リソースを、XtSetValues() 関数かリソース・ファイルのどちらかで設定します。XmNpreeditType() リソースは、TopLevelShell()、ApplicationShell()、またはDialogShell() ウィジェットのリソースとしても設定できます。この 3 つのウィジェットは、VendorShell() ウィジェット・クラスのサブクラスです。

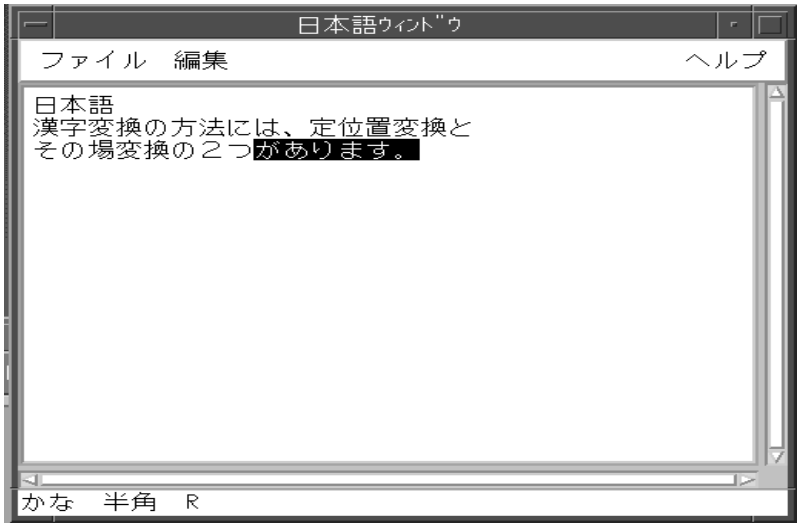

図 1–5　VendorShell ウィジェットでの OverTheSpot プリエディットの例 (日本語)

Root

Root モードでは、プリエディット領域およびステータス領域はクライアントのウィンドウとは別になっています。Root モードの動作は OffTheSpot に似ています。図 1–6 を参照してください。

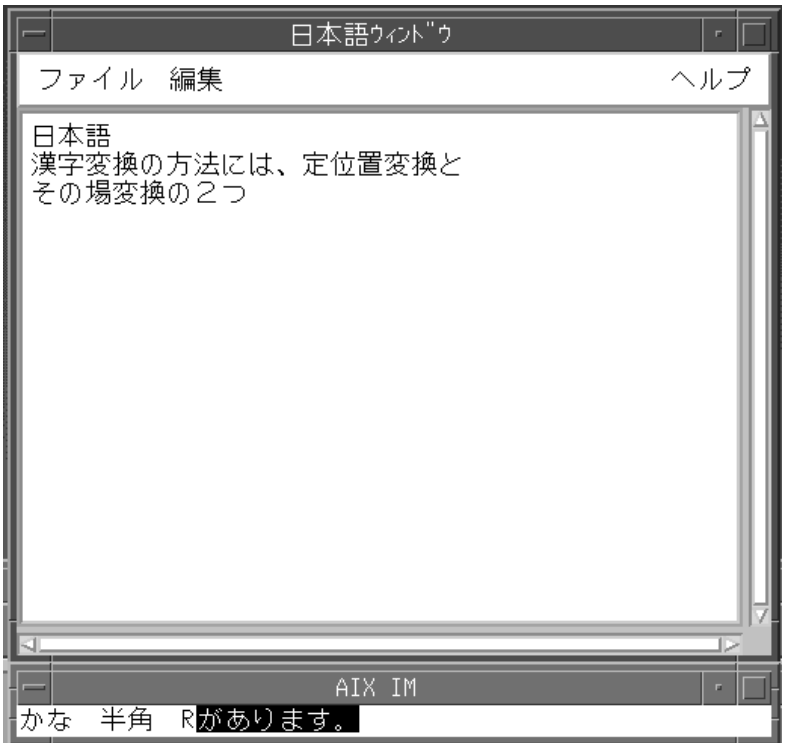

図 1–6　VendorShell ウィジェットでの Root プリエディットの例 (日本語)

ステータス領域

ステータス領域は、入力メソッドの入力ステータスまたはキーボード・ステータスをユーザに報告します。OverTheSpot および OffTheSpot の形式では、ステータス領域は VendorShell ウィンドウの左下隅に位置します。

- Root 形式の場合は、ステータス領域はクライアント・ウィンドウの外側に位置します。
- プリエディットの形式が OffTheSpot モードの場合は、プリエディット領域はステータス領域の右側に表示されます。

`VendorShell()` ウィジェットは、VendorShell ウィンドウがサイズ変更された場合にステータス領域が VendorShell ウィンドウの下隅に再配置できるように、ジオメトリ管理を提供しています。

補助領域

補助領域はユーザがプリエディットを行うときに役立ちます。補助領域は特定の入力メソッドにより作成できます。図 1–3 に示した日本語の入力メソッドは、次の型の補助領域を作成します。

- 全候補
- JIS コード
- 変換方法の切り替え
    - 先読み連文節
    - 一括連文節
    - 単文節
    - 複合語

メイン・ウィンドウ領域

メイン・ウィンドウ領域は、入力メソッドの作業対象の領域として使用されるウィジェットです。システム環境では、`VendorShell()` ウィジェットの子として作れるのは `MainWindow` ウィジェットだけです。`MainWindow` ウィジェットは、どんなコンテナ・ウィジェットだけにもなり得ます (`RowColumn()` ウィジェットなど)。ユーザはコンテナ・ウィジェットを `VendorShell()` ウィジェットの子として作成します。

フォーカス領域

フォーカス領域は、現在フォーカスを持っている `MainWindow()` ウィジェット・サブツリーの下のすべての子孫ウィジェットのうちの一つです。既存のウィジェットを使用している Motif アプリケーション・プログラマは、フォーカス領域を気にする必要はありません。重要なのは、一度に 1 つのウィジェットだけしか入力メソッドを処理できないということです。入力メソッドの処理は、現在フォーカスを持つウィンドウ (ウィジェット) に移動します。

クライアント間通信規約 (ICCC)

クライアント間通信規約 (ICCC) は、クライアント間でテキストを渡すのに使用する機構を定義します。システムは複数のコード・セットをサポートできるので、異なるコードセットを使用している 2 つのアプリケーションが互いに通信し合うことも可能です。ICCC は、2 つのクライアント間のデータの渡し方に関して、これらのクライアントがどのように同意するかを定義します。2 つのクライアントの持つ文字セットに互換性がない場合 (たとえばラテン 1 と日本語 (JIS) のように)、文字を転送するときにデータの一部が失われる可能性があります。

しかし、2 つのクライアントが、コード・セットは異なるが文字セットに互換性がある場合は、ICCC はこれらのクライアントがデータを失わずに情報を渡せるようにします。2 つのクライアントのコード・セットが等しくない場合は、COMPOUND_TEXT アトムを使用して、コンパウンド・テキストのエンコーディングが使用されます。通信中のデータにポータブルな文字 (7 ビット、ASCII、その他) または ISO8859-1 コード・セットしか含まれない場合は、データは XA_STRING アトムにより、変換なしでそのまま通信されます。

タイトル名とアイコン名は、ポータブルでない文字を使用する場合、COMPOUND_TEXT アトムを使用してウィンドウ・マネージャへ通信される必要があります。そうでない場合は、XA_STRING アトムを使用できます。その他のエンコーディングは、ウィンドウ・マネージャのロケールへ変換する程度に制限されています。ウィンドウ・マネージャは単一のロケールで実行され、実行中のロケールのコード・セットに変換できるタイトルとアイコン名しかサポートしません。

libXm ライブラリとすべてのデスクトップ・クライアントは、これらの規約に従う必要があります。

第 2 章

国際化対応と共通デスクトップ環境

1 つの共通オープン・システムには、異なる国語をサポートする複数の環境が存在することがあります。そのような 1 つ 1 つの国別環境はロケールと呼ばれ、言語、文字、フォント、入力やデータ・フォーマットの慣習を考慮します。共通デスクトップ環境は、どのアプリケーションでもシステム上にインストールされているすべてのロケールを使用して実行できるように、完全に国際化対応されています。

ロケール管理

デスクトップでは、ほとんどの単一画面のクライアントは、実行時に環境変数 (通常は $LANG) の設定から決定される単一のロケールで操作を行います。Xm ライブラリ (libXm) は、各ウィジェットがインスタンスを生成する際に使用する単一の

ロケールしかサポートできません。Xm ライブラリを初期化した後にロケールを変更した場合、予測できない動作をすることがあります。

すべての国際化対応プログラムは、ロケール環境変数に定義される、ユーザの希望するロケールを設定しなければなりません。デスクトップ・ツールキットを使用するプログラムの場合、プログラムはどのツールキット初期化関数 (たとえば `XtAppInitialize()`) よりも先に `XtSetLanguageProc()` 関数を呼び出します。この関数は、ツールキット初期化の前に必要なすべての初期化を行います。非デスクトップ・プログラムの場合は、プログラムはユーザの希望するロケールを設定するために、プログラムの最初に `setlocale()` 関数を呼び出します。

環境をコントロールするには、ロケール環境変数 (たとえば `LC_ALL`、`LC_CTYPE`、`LANG` など) が使用されます。ユーザは、実行時に使用されるロケール固有の機能を識別するために、ロケールの `LC_CTYPE` カテゴリが X および Xm ライブラリによって使用されることを覚えておいてください。しかし `LC_MESSAGES` カテゴリは、メッセージ・カタログ・サービスにより、ロケール固有のテキストを読み込むのに使用されます。詳細は、41ページの「ローカライズされたテキストの抽出」を参照してください。特にツールキットが読み込むフォントと入力メソッドは、`LC_CTYPE` カテゴリを設定することによって決定されます。

文字列のエンコーディング (たとえばアプリケーションのソースコード、リソース・ファイル、UIL (ユーザ・インタフェース言語) ファイルでの ISO8859-1 または 拡張 UNIX コード (EUC)) は、アプリケーションが実行するロケールのコード・セットと同じでなければなりません。同じでない場合はコード変換が必要です。

すべてのコンポーネントは単一で世界中で実行可能な状態で出荷され、ロケールの X11R5 サンプル・インプリメンテーション・セット (米国、西欧／東欧、日本、韓国、中国、台湾) をサポートする必要があります。

アプリケーションはコード・セットに依存せず、すべてのマルチバイト・コード・セットをサポートするように記述しなければなりません。

ロケール管理に使用する関数を次に示します。

- `XtSetLanguageProc()`
- `setlocale()`
- `XSupportsLocale()`
- `XSetLocaleModifiers()`

フォント管理

テキストを X Windows System のクライアントに受け渡す場合、少なくとも 2 つの状況が国際化対応に関係します。

- ローカライズされたテキスト自体を獲得すること
- ローカライズされたテキストの文字を描画するのに必要なすべてのグリフを含む 1 つ以上のフォントを選択すること

41ページの「ローカライズされたテキストの抽出」に、ローカライズされたテキストを獲得する方法が説明されています。

フォントを文字セットに一致させる

1 つのフォントには、ロケールの文字を描画するのに使用されるグリフのセットが含まれます。しかし、指定したロケールのために、次のような作業を実行する場合があります。

- 必要なフォントの決定
- 必要なフォントの指定
- リソース・ファイルでフォントの charset の決定
- 1 つのロケールに対して複数のフォントの選択

XLFD 名の最後の 2 つのフィールドは、フォントにどのグリフが含まれているか、セットから特定のグリフを獲得するのにどの値が使用されるかを識別します。これらの最後の 2 つのフィールドは、フォントに含まれるグリフのエンコーディングを識別します。

例

```
-adobe-courier-medium-r-normal--24-240-75-75-m-150-iso8859-1
```

この XLFD 名の最後の 2 つのフィールドは、`iso8859` と `1` です。これらのフィールドは、ISO8859-1 標準グリフがフォントに含まれるよう指定しています。さらに、ISO8859-1 標準の文字コード値を使用して、各文字に対応するグリフを索引付けするようにも指定しています。

アプリケーションがデータを受け渡すために使用するフォント charset は、選択するロケールによって決まります。フォント charset 情報はロケールの選択に基づく

ので、フォント指定はアプリケーションによってハードコードしてはいけません。その代わりに、ロケール固有の app-defaults ファイルにフォント指定を格納して、ローカライズされたバージョンの app-defaults ファイルを作成できます。

さらに、フォントはフォント・セットとして指定されます。フォント・セットは、複数のフォントを指定するのに XLFD が使用する Xlib の概念です。XLFD のフォント charset フィールドは、フォント・セットを作成して、ユーザが指定したロケールに基づいてこれらのフィールドを埋める Xlib コードで指定されます。

日本語、中国語、韓国語などの多くの言語で、単一のエンコーディングをサポートするのに複数の charset が組み合わされています。このような場合、文字を描画するためには必ず複数のフォントをオープンしなければなりません。さらに、データは各フォントに対応するセグメントへと解析されなければならず、場合によりそれらのセグメントの文字値をグリフ索引に変換しなければなりません。XFontset は、指定ロケールの文字データを描画するのに必要なすべてのフォントの集合ですが、このような問題も処理します。さらに、描画のためのセットとメトリック・ルーチンが提供されます。それらは内部で文字列を一貫した文字セットのセグメントに分け、値をグリフ索引に変換します。これらのルーチンはアプリケーション開発者の負担を軽減します。アプリケーション開発者は、ユーザのフォント・セットと新しい X11R5 描画とメトリック・アプリケーション・プログラム・インタフェース (API) だけを必要とします。

フォント・オブジェクト

この節では次のフォント・オブジェクトを説明します。

- フォント・セット
- フォント
- フォント・リスト

フォント・セット

一般的に、Xlib を使用してローカライズされたテキストを描画するすべての国際化対応プログラムは、ロケール依存フォントを指定するために XFontSet() を使用しなければなりません。1 つのフォント・セット内の特定のフォントは、charset フィールドには指定せずに、XLFD 命名規則を使って指定されます。XFontSet のリソース名は *fontSet です。フォント・リソースのリストは、45ページの「ローカライズされたリソース」を参照してください。

(XmString 関数やウィジェットを使用するのに対して) Xlib を直接使用してテキストを描画するアプリケーションは、Xt が提供する文字列からフォント・セットへのコンバータを利用できます。たとえば、次のコード・フラグメントは Xt を使用する場合と使用しない場合のフォント・セットの獲得方法を示します。

```
/* pardon the double negative... means "If using Xt..." */
#ifndef NO_XT
typedef struct {
        XFontSet fontset;
        char     *foo;
} ApplicationData,  *ApplicationDataPtr;
static XtResource my_resources[] = {
    { XtNfontSet, XtCFontSet, XtRFontSet, sizeof (XFontSet),
    XtOffset (ApplicationDataPtr, fontset), XtRString,
    "*-18-*"}}
#endif /* NO_XT */
...
#ifdef NO_XT
fontset = XCreateFontSet (dpy, "*-18-*", &missing_charsets,
    &num_missing_charsets. &default_string);
if (num_missing_charsets > 0) {
    (void) fprintf(stderr, "&s: missing charsets.\n",
                   program_name);
    XFreeStringList(missing_charsets);
}
#else
XtGetApplicationResources(toplevel, &data, my_resources,
                          XtNumber(my_resources), NULL, 0);
fontset = data.fontset;
#endif /* NO_XT */
```

フォント

国際化対応プログラムは、特定の charset と特定の文字セットのために使用する場合以外は、フォント (つまり XFontStruct) を直接使用することは避けてください。ロケールが必要とする特定の charset を接続しているサーバがサポートしていない場合、XFontStruct の使用は制限されます。XFontStruct のリソース名は *font です。

フォント・リスト

ローカライズされたテキストを描画するためにウィジェットや XmString を使用しているすべてのプログラムは、フォントを指定するために XmFontList 名を指定する必要があります。フォント・リストは 1 つ以上のフォント・セットまたはフォントのリストか、あるいはフォント・セットとフォント両方のリストです。フォント・リストは、ウィジェットがテキストを描画するために使用すべきフォントとフォント・セットのリストを指定するために使用されます。より複雑なアプリケーションの場合、フォント・リストは複数のフォント・セットを指定し、各フォン

ト・セットには名前がタグ付けされます (たとえば Bold、Large、Small など) 。このタグは XmString セグメントのタグと関連付けられます。タグは、フォント・リスト内の特定のフォントやフォント・セットを識別するのに使用されます。

フォント・セットおよびフォント・リストの形式

表 2–1 にフォント・セットとフォント・リストの形式を示します。

表 2–1 フォント・セットおよびフォント・リストの形式

リソース型	**XLFD** セパレータ	終止符	**FontEntry** セパレータ
*fontSet: (Xlib)	カンマ	なし	なし
*fontList : (Motif)	セミコロン	コロン	カンマ

次に、フォント・リソース指定の例をいくつか挙げます。

```
app_foo*fontList: -adobe-courier-medium-r-normal--24-240-75-75-m-150-*:
```

上記の fontList は、ユーザのロケールに適切なものとして、1 つ以上の 24 ポイント Adobe Courier フォントから成るフォント・セットを指定しています。

```
app_foo*fontList: -adobe-courier-medium-r-normal--18-*; *-gothic-*-18-*:
```

この fontList は、ユーザのデータの一部の文字には 18 ポイント Courier フォントから成るフォント・セットを指定し、その他の文字には 18 ポイント Gothic フォントを指定しています。

Motif ベースのアプリケーションは、フォント・リストに入っているフォント・セットに直接アクセスする必要が起こる場合があります。たとえば、DrawingArea ウィジェットを使用するアプリケーションは、描画したイメージの 1 つにラベルを付けたいことがあります。次のコード例は、フォント・リストからのフォント・セットの抽出方法を示しています。この例では、タグ XmFONTLIST_DEFAULT_TAG でフォント・セットを探します。XmFONTLIST_DEFAULT_TAG が「ロケールの codeset」を指すためです。アプリケーションでは、ローカライズされたデータを含むすべての文字列に XmFONTLIST_DEFAULT_TAG を使用してください。

```
XFontSet FontList2FontSet( XmFontList fontlist)
{
XmFontContext context;
```

```
XmFontListEntry next_entry;
XmFontType type_return = XmFONT_IS_FONT;
char* font_tag;
XFontSet fontset;
XFontSet first_fontset;
Boolean have_font_set = False;

if ( !XmFontListInitFontContext(&context, fontlist)) {
     XtWarning(``fl2fs: can't create fontlist context...'');
     exit 0;
}

while ((next_entry = XmFontListNextEntry(context) != NULL) {

  fontset = (XFontSet) XmFontListEntryGetFont(next_entry,
              &type_return);
  if (type_return == XmFONT_IS_FONTSET ) {

       font_tag = XmFontListEntryGetTag(next_entry);

       if (!strcmp(XmFONTLIST_DEFAULT_TAG, font_tag) {
           return fontset;
       }
       /* Remember the 1st fontset, just in case... */
       if (!have_font_set) {
           first_fontset = fontset;
           have_font_set = True;
       }
   }
}
if (have_font_set)
      return first_fontset;
return (XFontSet)NULL;
}
```

フォント関数

次の Xlib のフォント管理 API 関数を使用できます。

- `XCreateFontSet()`
- `XLocaleOfFontSet()`
- `XFontsOfFontSet()`
- `XBaseFontNameListOfFontSet()`
- `XFreeFontSet()`

次の Motif の FontList API 関数を使用できます。

- `XmFontListEntryCreate()`
- `XmFontListEntryAppend()`
- `XmFontListEntryFree()`
- `XmFontListEntryGetTag()`

- `XmFontListEntryGetFont()`
- `XmFontListEntryLoad()`

フォント **charset**

基本的な互換性を高めるために、フォントは標準 X コンソーシアムの font charset に沿って構成されています。

言語グループごとのデフォルト・フォント・セット

開発者の言語に関連付けられたフォント・セットのベース・フォント名を選択することは、通常は簡単です。開発者はその言語および必要なフォントのセットに慣れているからです。

しかし、さまざまなロケールのためのフォント・セットのベース・フォント名を選択する場合、XLFD フォント仕様は 15 ものフィールドから成るため、この作業は難しくなります。ローカライズする場合、フォント・セット選択のためには次のフィールドが重要です。

- FAMILY_NAME *%F*
- WEIGHT_NAME *%W*
- SLANT *%S*
- ADD_STYLE *%A*
- SPACING *%SP*

これによりフィールド数が減りますが、各フィールドの取り得る値はロケールごとに変化します。実際のポイント・サイズ (POINT_SIZE) は、プラットフォームごとに変化します。

このマニュアル全体を通して、ローカライズされたフォントを指定する際には次の規則が使用されます。

`-dt-%F-%W-%S-normal-%A-*-*-*-%SP-*`

以下に、リソース (`app-defaults`) ファイルにフォント・セットを指定する際に、デスクトップ内で使用する必須フィールドごとの推奨する最小セットを示します。

ラテン ISO8859-1 フォント

FOUNDRY	dt
FAMILY_NAME	interface user interface system application
WEIGHT_NAME	medium または bold
SLANT	r または i
ADD_STYLE	sans serif または serif
SPACING	p または m

その他の ISO8859 フォント

ISO8859-1 で定義するのと同じ値をお勧めします。

JIS 日本語フォント

FOUNDRY	dt
FAMILY_NAME	Gothic または Mincho
WEIGHT_NAME	medium または bold
SLANT	r
ADD_STYLE	*
SPACING	m

KSC 韓国語フォント

FOUNDRY	dt
FAMILY_NAME	Totum または Pathang
WEIGHT_NAME	medium または bold

SLANT	r
ADD_STYLE	*
SPACING	m

注 - FAMILY_NAME の値は、2 つの共通フォント・ファミリの公的なローマ字表記法のうち使用しているものによって変化します。背景として、Totum は通常、ゴシック、 Kodig、 または Dotum として出荷されるフォントに対応します。Pathang は通常 Myungo か Myeongjo として出荷されるフォントに対応します。

CNS 繁体字中国語フォント

FOUNDRY	dt
FAMILY_NAME	Sung と Kai
WEIGHT_NAME	medium または bold
SLANT	r
ADD_STYLE	*
SPACING	m

GB 簡体字中国語フォント

FOUNDRY	dt
FAMILY_NAME	Song と Kai
WEIGHT_NAME	medium または bold
SLANT	r
ADD_STYLE	*
SPACING	m

ローカライズされたテキストの描画

ローカライズされた文字列を描画するための機構がいくつか提供されていますが、Motif ライブラリが使用されているか Xlib ライブラリが使用されているかに依存します。以下では、国際化対応アプリケーションに対してお勧めできるインタフェースを説明します。しかし、ローカライズされたデータはすべて、シンプル・テキストを使用するプログラムの外側に置くことをお勧めします。

シンプル・テキスト

次の Xlib マルチバイト (`char*`) 描画関数は国際化対応しています。

- `XmbDrawImageString()`
- `XmbDrawString()`
- `XmbDrawText()`

次の Xlib ワイド文字 (`wchar_t*`) 描画関数は国際化対応しています。

- `XwcDrawImageString()`
- `XwcDrawString()`
- `XwcDrawText()`

次の Xlib マルチバイト文字 (`char*`) フォント・メトリック関数は国際化対応しています。

- `XExtentsOfFontSet()`
- `XmbTextEscapement()`
- `XmbTextExtents()`
- `XmbTextPerCharExtents()`

次の Xlib ワイド文字 (`char_t*`) フォント・メトリック関数は国際化対応しています。

- `XExtentsOfFontSet()`
- `XwcTextEscapement()`
- `XwcTextExtents()`
- `XwcTextPerCharExtents()`

XmString (コンパウンド・ストリング)

Xm ライブラリの場合、ローカライズされたテキスト
は、XmStringCreateLocalized()を使用して XmString セグメントに挿入しなければなりません。ローカライズされたテキストに関連付けられたタグは XmFONTLIST_DEFAULT_TAG であり、これはフォント・リストのエントリに一致させるのに使用されます。XmStringCreate() を使用してコンパウンド・ストリング内でいくつかのフォントを混合するアプリケーションは、ローカライズされたすべての文字列に対してタグとして XmFONTLIST_DEFAULT_TAG を使用してください。

さらに重要なのは、クライアント間通信の場合、XmStringConvertToCT() 関数が XmFONTLIST_DEFAULT_TAG とタグ付けされたセグメントを、ロケールのコード・セットにエンコードされるよう関連付けることです。そうしない場合、使用されるタグ名によっては、Xm ライブラリはテキスト・データのクライアント間通信のエンコードを正しく識別できないことがあります。

XmString 内のローカライズされた文字列セグメント
は、XmFONTLIST_DEFAULT_TAG の付いたフォント・セットを持つフォント・リストで描画することができます。移植性をよくするために、ローカライズされた文字列を使用してください。

次に、ローカライズされた文字列を描画するためにフォント・リストを作成する例を示します。

```
XmFontList CreateFontList( Display* dpy, char* pattern)
{

SmFontListEntry font_entry;
XmFontList fontlist;
font_entry = XmFontListEntryLoad( dpy, pattern,
                                        XmFONT_IS_FONTSET,
                                        XmFONTLIST_DEFAULT_TAG);

fontlist = XmFontListAppendEntry(NULL, font_entry);
/* XmFontListEntryFree(font_entry); */

if ( fontlist == NULL ) {
     XtWarning(``fl2fs: can't create fontlist...'');
     exit (0);
}

return fontlist;
}

int main(argc,argv)
int argc;
char **argv;
{
    Display *dpy;            /* Display                */
    XtAppContext app_context;/*  Application Context    */
```

```
    XmFontList fontlist;
    XmFontSet fontset;
    XFontStruct** fontstructs;
    char** fontnames;
    int i,n;

char *progrname;     /* program name without the full pathname */

if (progname=strrchr(argv[0], '/')){
    progname++;
}
else  {
    progname = argv[0];
}

/*    Initialize toolkit and open display.
 */
XtSetLanguageProc(NULL, NULL, NULL);
XtToolkitInitialize():
app_context = XtCreateApplicationContext();
dpy = XtOpenDisplay(app_context, NULL, progname, ''XMdemos'',
                      NULL, 0, &argc, argv);
if (!dpy) {
      XtWarning(''fl2fs: can't open display, exiting...'');
      exit(0);
}

fontlist = CreateFontList(dpy, argv[1] );
fontset = FontList2FontSet( fontlist );

/*
 * Print out BaseFontNames of Fontset
 */
n = XFontsOfFontSet( fontset, &fontstructs, &fontnames);

     printf(''Fonts for %s is %d\n'', argv[1], n);

     for (i = 0 ; i < n ; ++i ) printf(''font[%d} - %s\n'', i,\
                                        fontnames[i] );
     exit(1);
}
```

リソース・ファイルに指定されたコンパウンド・ストリング
は Xm_FONTLIST_DEFAULT_TAG の付いたロケール・エンコーディング・セグメントを持っているため、ローカライズされた文字列はリソース・ファイルに記述できます。たとえば、次の例の fontList リソースは、自動的に XmFONTLIST_DEFAULT_TAG に関連付けられます。

```
labelString: Japanese string
 *fontList: -dt-interfacesystem-medium-r-normal-L*-*-*-*-*-*-*:
```

国際化対応には、次の XmString 関数のセットをお勧めします。

- XmStringCreateLocalized()
- XmStringDraw()

- XmStringDrawImage()
- XmStringDrawUnderline()

次の XmString() 関数のセットは、サポートされていない言語では動作しない可能性があることを示す情報を使用するので、国際化対応にはお勧めしません。

- XmStringCreateLtoR()
- XmStringSegementCreate()

ローカライズされたテキストの入力

ローカライズされたテキストの入力は、通常、ローカルな入力メソッドかネットワークを使った入力メソッドのどちらかを使用して行います。

ローカルな入力メソッドは、入力メソッドが Xlib 内に実装されることを意味します。通常この方法は、単純な規則で構成できて言語固有の機能を必要としない言語で使用します。ネットワークを使った入力メソッドは、実際の入力メソッドが別のサーバで提供され、Xlib は言語固有の構成を行うために XIM プロトコルを介してそれらのサーバと通信することを意味します。

基本的な入力要求およびダイアログ

アプリケーションですべてのテキスト入力を行うのに Text ウィジェットを使用することを強くお勧めします。

DrawingArea ウィジェット内の入力

多くのアプリケーションは、入力に基づくウィジェットの中で描画を行います。デスクトップ環境内での一貫性を提供するために、XmIm 関数をお勧めします。これは、入力メソッドに必要な形式およびジオメトリ管理が VendorShell ウィジェット・クラスで管理されるからです。アプリケーションに関係するのは、キー・イベント、フォーカス、描画領域内の現在の入力位置の通信だけです。XmIm 関数を使用するには下位の Xlib 入力メソッドのアーキテクチャの基本知識が必要ですが、開発者は XmIm の情報を理解していれば問題ありません。

アプリケーション固有および言語固有の中間フィードバック

一部のアプリケーションは、プリエディットにおける中間フィードバックを直接表示する必要があります。たとえばアプリケーションが Xlib の機能の限度を超えてしまうことがあります。この例には、PostScript™ による描画や縦書きの仕様が挙げられます。

コア Xlib は、アプリケーションがプリエディットにおける中間フィードバックを表示できるようにするインタフェースの共通セットを提供します。アプリケーションのコールバックを登録し、プリエディットの形式を `XNPreeditCallbacks` に設定することで、アプリケーションは入力メソッドから中間のプリエディット・データを獲得し、必要なものをすべて描画できます。

複雑な言語処理を行うアプリケーションは、特定の XIM 処理系とその入力エンジンの中の拡張機能を認識することがあります。そのようなアプリケーションは最先端であり、XIM 関数の詳しい知識と経験を必要とします。

Text ウィジェットおよび TextField ウィジェット

基本プロンプトおよびダイアログには、`Text` ウィジェットか `TextField` ウィジェットをお勧めします。`Text[Field]` ウィジェットの中にローカライズされたテキストを獲得および設定するのに、リソースの他にすべての `XmTextField` 関数と `XmText` 関数を使用できます。

ほとんどの `XmText` 関数は、バイト数ではなく文字数に基づいています。たとえば、すべての `XmTextPosition()` 関数位置は文字の位置であり、バイトの位置ではありません。`XmTextGetMaxLength()` 関数はバイト数を返します。疑わしい場合、位置は常に文字単位であることを思い出してください。

`Text` ウィジェットまたは `TextField` ウィジェットの幅は、`XmNcolumns` のリソース値によって決定します。しかし、この値はフォント・セットの最も幅の広い文字の数を表すものであり、バイト数やカラム数ではありません。たとえば、`Text` ウィジェットに可変幅のフォントを選択したとします。文字 *i* の幅は 1 ピクセルであり、文字 *W* の幅は 7 ピクセルです。`XmNcolumns` の値が 10 に設定されると、これは `Text` ウィジェットを 10 文字以上表示できる幅にするための要求と見なされます。したがって `Text` ウィジェットは、最も広い文字の幅を使って `Core` ウィジェットのピクセル幅を決定しなければなりません。この例では、ウィジェットに *W* なら 10 個、*i* なら 70 個表示できます。`XmNcolumns` のこの構造は、コード・セットがマ

ルチバイトでマルチカラム・エンコーディングのロケールの場合、問題を起こす可能性があります。この値はローカライズされたリソースの中で設定してください。

次節は、入力メソッドの管理に使用するアプリケーションで使用可能な関数のセットを示します。Text ウィジェットおよび TextField ウィジェットを使用するアプリケーションについては、51ページの「入力メソッド (キーボード)」を参照してください。

Text[Field] ウィジェットを使用しないカスタマイズされたウィジェット内での文字入力

アプリケーションはユーザから文字入力を受け付けても、そのために TextField ウィジェットや Text ウィジェットを使用しない場合があります。たとえば、DrawingArea ウィジェットを使用するアプリケーションでは、ユーザは直接 DrawingArea にテキストを入力できます。この場合、アプリケーションは、後の節で説明するように Xlib XIM 関数を使用できます。または、代わりにアプリケーションは Motif 1.2 の XmIm 関数を使用できます。XmIm 関数により、アプリケーションは最小限のコードで入力メソッドに接続および対話できます。さらに、XmIm 関数によって Motif VendorShell ウィジェットがアプリケーションに代わって入力メソッドのジオメトリ管理を実行できます。

XmIm 関数は Motif 1.2 のすべての処理系に含まれて出荷されますが、XmIm 関数は Motif 1.2 のドキュメントには記載されていません。OSF は、Motif 2.0 のために XmIm 関数を増やしてドキュメント化する意向を明らかにしています。ここで説明する関数は Motif 1.2 の XmIm 関数です。

注 - Motif 1.2 の XmIm 関数は、プリエディット・コールバック形式やステータス・コールバック形式の入力メソッドをサポートしません。Xlib API によりプリエディット・コールバックが使用できます。詳細は、38ページの「XIM 管理」を参照してください。

Motif 1.2 ベースのアプリケーションで安全に使用できる XmIm 関数を次に示します。パラメータと型の正式な説明は xm.h ヘッダ・ファイルにあります。

関数名	説明
XmImRegister()	XOpenIM() を実行し、サポートされている形式の入力メソッドを照会します。
XmImSetValues()	プリエディットおよびステータスの形式を調整および選択します。
XmImSetFocusValues()	XIC がなければ作成します。入力メソッドにウィジェットがフォーカスを獲得したことを通知します。XIC に渡される値を設定します。
XmImUnsetFocus()	入力メソッドにウィジェットがフォーカスを失ったことを通知します。
XmImMbLookupString()	XmbLookupString() と等価の Xm の関数で、1 つ以上のキー・イベントを文字に変換します。戻り値は XmbLookupString() と等価です。
XmImUnregister()	入力メソッドとウィジェットを切り離し、新しい入力メソッドへの接続を可能にします。必ずしも入力メソッドを閉じるわけではありません (処理系によります)。

XmImSetValues() 関数と XmImSetFocusValues() 関数により、アプリケーションは入力メソッドが必要とする情報を渡すことができます。すべての値が必要ではない場合でも、(それぞれの値はプリエディットおよびステータスの形式をサポートするので) アプリケーションがすべての値を渡すことが重要です。これは、ユーザまたは VendorShell ウィジェットによってどの形式が選択されたかをアプリケーションは認識できないからです。次に、XmImSet[Focus]Values() 関数への呼び出しで渡されるべき各値の引き数とデータ型を示します。

引き数名	データ型
XmNbackground	Pixel
XmNforeground	Pixel
XmNbackgroundPixma	Pixmap
XmNspotLocation	XPoint

引き数名	データ型
XmNfontList	Motif fontlist
XmNlineSpace	int (連続ベースライン間のピクセルの高さ)

XmIm 関数は次のように使用されます。

- ツールキットを初期化する前に、アプリケーションはロケールを初期化するために XtSetLanguageProc(NULL, NULL, NULL) を呼び出します。
- 文字入力をしたい箇所でウィジェットを作成した後、アプリケーションは入力メソッドを開いて接続を確立するために XmImRegister(widget) を呼び出します。
- 入力メソッドへの接続を確立した後、アプリケーションは XmImSetValues() を呼び出してリストされているすべての値を渡すことにより、初期 XIC 値を入力メソッドに渡します。この関数は引き数 arg_list と number_args を取ります。arglist は XtSetArg() を呼び出すことで設定されます。
- XtAddEventHandler() 関数を介して、入力メソッドから入力を獲得するウィジェットのマネージャ・ウィジェットに対してイベント・ハンドラを追加します。イベント・ハンドラは FocusChangeMask マスク用です。このハンドラは、フォーカスを獲得したときに XmImSetFocusValues() を呼び出し、フォーカスを失ったときに XmImUnsetFocus() を呼び出します。入力メソッドのためのフォーカス設定するとき、上記にリストされた値の完全なセットを渡します。
- 入力メソッドから入力を獲得するウィジェットに対して DestroyCallback を追加します。デストロイ・コールバックでは、ウィジェットと入力メソッドの間の接続を切り離していることを入力メソッドに通知するために、XmImUnregister() を呼び出します。
- 上記にリストされた 1 つ以上の入力メソッド値に変更があった時 (たとえば spotLocation) に入力メソッドに通知するため、XmImSetValues() を使用します。

XIM 管理

XIM 管理関数を次に示します。

関数名	説明
XOpenIM()	入力メソッドへの接続を確立します。
XCloseIM()	あらかじめ XOpenIM() への呼び出しで確立された入力メソッドへの接続を閉じます。
XGetIMValues()	属性リストの入力メソッドを照会します。現在、Xlib での唯一の標準引き数は XNQueryInputstyle です。
XDisplayOfIM()	入力メソッドに関連付けられたディスプレイを返します。
XLocaleOfIM()	入力メソッドのロケールを識別する文字列を返します。標準的な文字列はありません。この呼び出しで返される値は処理系が定義します。
XCreateIC()	入力コンテキストを作成します。入力コンテキストには、(もしあれば) 入力メソッドが必要とするデータと、そのデータを表示するのに必要な情報の両方が含まれます。
XDestroyIC()	入力メソッドを破棄し、関連付けられていたメモリを解放します。
XIMOfIC()	指定された入力コンテキストに、現在関連付けられている入力メソッドを返します。
XSetICValues()	文字データの入力、プリエディット情報、またはステータス情報の表示をコントロールするために、0 個以上の値を入力コンテキストに渡します。すべての有効な入力コンテキスト値の引き数の表が、X11R5 の仕様書に記述されています。
XGetICValues()	0 個以上の入力コンテキスト値を得るために入力コンテキストを照会します。すべての有効な入力コンテキスト値の引き数の表が、X11R5 の仕様に記述されています。

XIM イベント処理

XIM イベント処理関数を次に示します。

関数名	説明
XmbLookupString()	キーを押したイベントをマルチバイト文字に変換します。
XwcLookupString()	キーを押したイベントをワイド文字に変換します。
XmbResetIC()	入力コンテキストを初期状態にリセットします。そのコンテキストで保留されている入力はすべて削除されます。現在のプリエディットの値を char* 文字列として返します。入力メソッドの処理系によっては、戻り値は NULL になります。
XwcResetIC()	入力コンテキストを初期状態にリセットします。そのコンテキストで保留されている入力はすべて削除されます。現在のプリエディットの値を wchar_t* 文字列として返します。
XFilterEvent()	クライアントへのすべての着信イベントを、アプリケーションが処理する前に、入力メソッドが処理できるようにします。
XSetICFocus()	指定された入力コンテキストに接続されたフォーカス・ウィンドウが、キーボード・フォーカスを受信したことを入力メソッドに通知します。
XUnsetICFocus()	指定された入力コンテキストがキーボード・フォーカスを失い、そのコンテキストに接続されたフォーカス・ウィンドウには入力できなくなったことを入力メソッドに通知します。

XIM コールバック

X 入力メソッド (XIM) は 3 種類のコールバックを提供します。1 番目はプリエディット・コールバックで、これによりアプリケーションはプリエディットの間に中間フィードバックを表示できます。2 番目はジオメトリ・コールバックで、これによりアプリケーションと XIM は XIM で使用されるジオメトリをネゴシエートできます。3 番目はステータス・コールバックで、これによりアプリケーションは XIM の内部ステータスを表示できます。

表 2–2 XIM Callbacks

XIM プリエディット・コールバック	XIM ステータス・コールバック	XIM プリエディット・キャレット・コールバック	XIM ジオメトリ・コールバック
PreeditStartCallback()	StatusStartCallback()	PreeditCaretCallback()	GeometryCallback()
PreeditDoneCallback()	StatusDoneCallback()		
PreeditDrawCallback()	StatusDrawCallback()		

ローカライズされたテキストの抽出

アプリケーションをローカライズする方法はいくつかありますが、一般的な規則では、言語依存情報はアプリケーションの外側に置き、ロケール名で識別される別のディレクトリに格納します。

この節では、アプリケーションの言語環境を確立するために、ユーザとアプリケーション開発者と処理系の組み合わせ方を説明します。アプリケーションをローカライズするための 2 つの一般的なアプローチについても説明します。次の 3 つの方法があります。

- リソース・ファイル
- メッセージ・カタログ
- プライベート・ファイル

リソース・ファイル

リソース・ファイルは、アプリケーションに関するあらゆる種類の情報をカスタマイズするための GUI ツールキット機構です。イントリンシクス・ライブラリ (libXt) が、コマンド行オプション、アプリケーション定義のリソース、ユーザ定義のリソースをマージするための高性能な機構を提供します。リソース・ファイルは、ローカライズされたテキストの抽出にも使用できます。リソース・ファイルとメッセージ・カタログの違いは、リソース・データベースは読み込まれるたびに毎

回コンパイルされることです。したがって、どの文字列をリソース・ファイルに置き、どの文字列をメッセージ・カタログに置くかは注意して決定してください。

また、Xm ライブラリ関数は、ローカライズされたリソースが読み込まれる位置を指定するときに、LC_MESSAGES カテゴリには依存しないので注意してください。詳細は、`XtSetLanguageProc()` のマニュアル・ページを参照してください。

メッセージ・カタログ

メッセージ・カタログは、ローカライズされたテキストを含む外部データベースにアクセスするための、伝統的なオペレーティング・システムの機構です。これらの関数は、あらかじめコンパイルされてアクセスの準備ができているカタログ・ファイルを読み込みます。またこれらは、カタログが見つからない場合のために、実際のプログラム内にデフォルトのメッセージを用意しています。

メッセージ機能のサポートは XPG4 と System V Release 4 (SVR4) の両方のメッセージ・カタログへのアクセスのインタフェースに基づいています。

プライベート・ファイル

ローカリゼーション・テキストだけでなく、カスタマイズされた総称的なデータベースを提供するために、アプリケーションはプライベート・データベースを使用できます。このようなデータベースは、通常はテキストを格納します。データベースが多数のファイルに広がるようであれば、ローカライズされたテキストの実行時の間接アクセスが提供されます。このアクセスがないと、ローカリゼーションは普通のユーザにとっては難しくなります。一般的に、このようなプライベート・ファイル形式はローカリゼーションを行うグループに反対されます。しかし、テキストのみのローカリゼーション専用のツールが提供されれば問題は少なくなります。

メッセージ・ガイドライン

メッセージ・ガイドラインは、メッセージおよびヘルプ情報を一貫性のある形式にするのに役立ちます。また、経験の少ない翻訳者はもちろん、英語に堪能でないエンド・ユーザにも簡単に理解できるメッセージの作成と管理を促進します。これらのガイドラインを利用して、一貫性のある言葉で意味が明確にわかるメッセージ・ファイルを作成してください。これらのガイドラインが配布されることによって、

プログラマと記述者がメッセージの記述に関して整合性を取ることができます。国際化対応サポートを完全に実現するには、各実行ファイルに対してデフォルト・メッセージ、外部メッセージ・ファイル、翻訳可能なメッセージの計画的な配布が必要です。

メッセージ抽出関数

国際化対応を行うプログラム (基本コマンドおよびユーティリティを含む) の要求事項の 1 つは、出力デバイスに表示されるメッセージがユーザの言語であることです。これらのプログラムは多くの国々 (国際的なロケール) で使用されるため、メッセージはそれらの国々のさまざまな言語に翻訳されなければなりません。

デスクトップ環境には、XPG4 関数と Xlib 関数の 2 つのメッセージ抽出関数のセットがあります。

XPG4/ 統合 UNIX メッセージ関数

XPG4 メッセージ機能は、メッセージ・ソース・ファイル、カタログ生成機能、プログラミング・インタフェースから構成されています。次に示すのは、XPG4/ 統合 UNIX メッセージ関数です。

- `catopen()`
- `catgets()`
- `catclose()`

XPG4 メッセージ例

次の例はカタログからメッセージを取り出す方法を示しており、3 つのパートがあります。1 番目のパートはメッセージ・ソース・ファイルを示し、2 番目のパートはカタログ・ファイルを生成するために使う方法を示します。3 番目のパートは、そのカタログを使用したプログラム例です。

メッセージ・ソース・ファイル

メッセージ・カタログは次のように指定します。

```
example.msg ファイル
$quote ''
$ every message catalog should have a beginning set number.
$set 1 This is the set 1 of messages
1 ''Hello world\n''
2 ''Good Morning\n''
3 ''example: 1000.220 Read permission is denied for the file
%s.\n''
$set 2
1 ''Howdy\n''
```

カタログ・ファイルの生成

このファイルは、メッセージ・カタログ example.cat を生成するために gencat ユーティリティに次のように入力されます。

```
gencat example example.msg
```

プログラム内のカタログへのアクセス

```
#include <locale.h>
#include <nl_types.h>
char *MF_EXAMPLE = "example.cat"

main()
{
        nl_catd catd;
        int error;

        (void)setlocale(LC_ALL, ''');

        catd = catopen(MF_EXAMPLE, 0);
            /* Get the message number 1 from the first set.*/

        printf( catgets(catd,1,1,''Hello world\n'') );
            /* Get the message number 1 from the second set.*/

        printf( catgets(catd, 2, 1,''Howdy\n'') );
            /* Display an error message.*/

        printf( catgets(catd, 1, 4,''example: 100.220
                Permission is denied to read the file %s.\n'') ,
                MF_EXAMPLE);
        catclose(catd);
}
```

Xlib メッセージ関数

次の Xlib メッセージ関数は、リソースに対して同じような入力／出力 (I/O) 操作を提供します。

- XrmPutFileDatabase()
- XrmGetFileDatabase()
- XrmGetStringDatabase()

- `XrmLocaleOfDatabase()`

これらは『*X Window System, The Complete Reference to Xlib, Xprotocol, ICCCM, XLFD - X Version 11, Release 5*』に説明されています。

Xlib メッセージおよびリソース機能

システム環境の国際化対応の一部であるツールキット・ベースのアプリケーションは、アプリケーションのソースの中にハードコードされたロケール固有のデータを持っていません。ロケール固有の共通項目は、標準 I/O のアプリケーションによって返される (エラーおよび警告) メッセージです。

一般に、システム環境ツールキット・ウィジェットまたはガジェットを介してユーザに表示されるすべてのエラー・メッセージおよび警告メッセージについては、メッセージ・カタログを介してメッセージをアプリケーションの外側に置いてください。

ツールキット・コンポーネントを介して表示されるダイアログ・メッセージについては、ローカライズされたリソース・ファイルを介してメッセージをアプリケーションの外側に置いてください。この方法は、`XmLabel` と `XmPushButton` クラスの `XmNlabelString` リソースやウィンドウのタイトルなどのリソースのローカライズと同じです。

たとえば、警告メッセージが `XmMessageBox` ウィジェット・クラスを介して表示される場合、`XmNmessageString` リソースはアプリケーションのソースコード内でハードコードできません。その代わりに、このリソースの値はメッセージ・カタログから取り出さなければなりません。異なる複数のロケールで実行される国際化対応のアプリケーションの場合は、サポートされる各ロケールに対してローカライズされた別個のカタログが存在しなければなりません。このようにすれば、アプリケーションは再構築する必要はありません。

ローカライズされたリソース・ファイルは `/usr/lib/X11/%L/app-defaults` サブディレクトリに入るか、または `XENVIRONMENT` 環境変数によって指定されます。`%L` 変数は、実行時に使用されるロケール名に置換されます。

ローカライズされたリソース

この節では、どのウィジェットおよびガジェット・リソース内容がロケールに依存するかを説明します。情報は関連機能別に構成されています。たとえば、最初のセ

クションでは、ラベルを表示したりプッシュ・ボタン機能を提供するのに使用されるウィジェットのためのロケールを区別するリソースを説明します。

ラベルおよびボタン

表 2–3 は、ラベルとして使用されるローカライズできるリソースをリストしています。多くは XmString 型です。その他の型は color か char* です。これらのリソースの詳しい説明は、『*Motif 1.2 Reference Manual*』を参照してください。どんな場合でも、アプリケーションはこれらのリソースをハードコードしてはいけません。リソース値がアプリケーションによって指定されなければならない場合は、app-defaults ファイルで指定してください。それにより、そのリソースがローカライズできることが保証されます。

ここにはウィジェット・クラス・リソースのみを示し、それらのウィジェットのサブクラスは示していません。たとえば、XmDrawnButton ウィジェット・クラスは、ローカライズされた新しいリソースを使えるようにしませんが、このウィジェットは XmLabelWidget ウィジェット・クラスのサブクラスです。したがって、そのアクセラレータ・リソース、acceleratorText リソースその他が同様にローカライズされ、アプリケーションによるハードコードはできません。

表 2–3 ローカライズできるリソース

ウィジェット・クラス	リソース名
Core	*background: [1]
XmCommand	*command:
XmCommand	*promptString:
XmFileSelectionBox	*dirListLabelString:
XmFileSelectionBox	*fileListLabelString:
XmFileSelectionBox	*filterLabelString:
XmFileSelectionBox	*noMatchString:
XmLabel[Gadget]	*accelerator:

表 2–3 ローカライズできるリソース　続く

ウィジェット・クラス	リソース名
XmLabel[Gadget]	*acceleratorText:
XmLabel[Gadget]	*labelString:
XmLabel[Gadget]	*mnemonic:
XmList	*stringDirection:
XmManager	*stringDirection:
XmMessageBox	*cancelLabelString:
XmMessageBox	*helpLabelString:
XmMessageBox	*messageString:
XmMessageBox	*okLabelString:
XmPrimitive	*foreground: [1]
XmRowColumn	*labelString:
XmRowColumn	*menuAccelerator:
XmRowColumn	*mnemonic:
XmRowColumn(SimpleMenu*)	*buttonAccelerators:
XmRowColumn	*mnemonic:
XmRowColumn	*mnemonic:
XmRowColumn	*mnemonic:
XmRowColumn	*mnemonic:
XmSelectionBox	*applyLabelString:

表 2–3　ローカライズできるリソース　続く

ウィジェット・クラス	リソース名
XmSelectionBox	*cancelLabelString:
XmSelectionBox	*helpLabelString:
XmSelectionBox	*listLabelString:
XmSelectionBox	*okLabelString:
XmSelectionBox	*selectionLabelString:
XmSelectionBox	*textAccelerators:

1. カラー名はポータブル文字セットに限定するという X プロトコルの制限により、フォアグラウンド・カラーおよびバックグラウンド・カラーはローカライズされません。ローカライズされたカラー名の提供方法は、アプリケーションにゆだねられ、ポータブル文字セットでエンコードされた名前にマップするためにローカライズされたデータベースを提供するべきです。

XmRowColumn ウィジェットには、別のローカライズ可能な文字列リソースがあります。それらのリソースは XmRowColumn のマニュアル・ページの、「シンプル・メニュー作成リソース・セット」の見出しの所にリストされています。このタイトルが示すように、これらのリソースは XmCreateSimpleMenu() 関数で作成された RowColumn ウィジェットにしか影響を与えません。影響を受けるリソースは次のとおりです。

*buttonAccelerator、*buttonAcceleratorText、*buttonMnemonics、
*optionLabel、*optionMnemonic

これらのリソースはほとんど使用されず、またシンプル・メニューの作成時のみ RowColumn に適用されるので、表 2–3 には入っていません。

リスト・リソース

いくつかのウィジェットにより、アプリケーションはウィジェットの項目のリストを設定または読み込むことができます。表 2–4 は、どのウィジェットによってそれ

を実行できるか、またウィジェットがリストを設定または読み込むのに使用するリソースを示します。リスト項目はローカライズされるかもしれないので、それらのリストをハードコードしないでください。それよりは、app-defaults ファイルにリソースとして設定し、ローカライズできるようにしてください。各リストの型は XmStringList です。

表 2–4 リストの読み込みに使用されるリソース

ウィジェット・クラス	リソース名
XmList	*items:
XmList	*selectedItems:
XmSelectionBox	*listItems:

タイトル

表 2–5 はタイトルおよびアイコン名の設定に使用されるリソースの一覧です。通常、アプリケーションが設定する必要があるのは *title: リソースと *iconName: リソースだけです。それぞれのエンコーディングは、適切なロケール管理を行なっているクライアントでは自動的に検出されます。これらの型はすべて char * または XmString です。

表 2–5 タイトルおよびアイコン名の設定に使用されるリソース

ウィジェット・クラス	リソース名
TopLevelShell	*iconName:
TopLevelShell	*iconNameEncoding: [1]
WmShell	*title:
WmShell	*titleEncoding: [1]

表 2–5　タイトルおよびアイコン名の設定に使用されるリソース　続く

ウィジェット・クラス	リソース名
XmBulletinBoard	*dialogTitle:
XmScale	*titleString:

1. このリソースはアプリケーションが設定してはなりません。アプリケーションが XtSetLanguageProc を呼び出すと、このリソースのデフォルト値 (なし) が自動的に設定され、それにより、ローカライズされたテキストがタイトルに使用できることが保証されます。

Text ウィジェット

表 2–6 は、ロケール別に設定する、または国際化対応アプリケーションの開発者が知っておくべき Text[Field] リソースの一覧です。

表 2–6　ロケール別に設定する Text[Field] リソース

ウィジェット・クラス	リソース名
XmSelectionBox	*textColumns: [1]
XmSelectionBox	*textString:
XmText	*columns: [1]
XmText	*modifyVerifyCallback:
XmText	*modifyVerifyCallbackWcs:
XmText	*value:
XmText	*valueWcs:
XmTextField	*columns: [1]

表 2–6　ロケール別に設定する Text[Field] リソース　続く

ウィジェット・クラス	リソース名
XmTextField	*modifyVerifyCallback:
XmTextField	*modifyVerifyCallbackWcs:
XmTextField	*value:
XmTextField	*valueWcs:

1. *columns リソースは、表示される文字数によって Text[Field] ウィジェットの初期幅を指定します。可変幅のフォントの場合、また文字サイズが非常に変化するロケールの場合は、カラムはそのロケールの文字レパートリーの中で最も幅の広い文字を表示するのに必要なスペースの量です。たとえば、カラム幅が 10 なら、現在のロケールの文字が少なくとも 10 文字表示できることが保証されます。割り当てられたスペースに、その数以上の文字数を表示することも可能です。

入力メソッド (キーボード)

表 2–7 は、入力メソッドのカスタマイズのためにローカライズできるリソースの一覧です。それらのリソースにより、ユーザまたはアプリケーションは、指定されたロケールにどの入力メソッドが使用されるか、また (プリエディットが適用でき、使用可能な場合) どのプリエディット形式が使用されるかを制御できます。

表 2–7　入力メソッドのカスタマイズ用のローカライズできるリソース

ウィジェット・クラス	リソース名
VendorShell	*inputMethod:
VendorShell	*preeditType:

ピックスマップ (アイコン)・リソース

表 2–8 はピックスマップ・リソースの一覧です。場合により、指定されたロケールに対して違うピックスマップが必要になる場合があります。

表 2–8 ピックスマップ・リソース

ウィジェット・クラス	リソース名
Core	*backgroundPixmap:
WMShell	*iconPixmap:
XmDragIcon	*pixmap:
XmDropSite	*animation[Mask|Pixmap]:
XmLabel[Gadget]	*labelInsensitivePixmap:
XmLabel[Gadget]	*labelPixmap:
XmMessageBox	*symbolPixmap:
XmPushButton[Gadget]	*armPixmap:
XmToggleButton[Gadget]	*selectInsensitivePixmap:
XmToggleButton[Gadget]	*selectPixmap:

ピックスマップは、必要なときに再呼び出しして表示できるようにメモリに格納された画面イメージです。デスクトップには多数のピックスマップ・リソースがあり、それによってアプリケーションは、バックグラウンド、ボーダ、影、ラベルとボタンの表面、ドラッグ・アイコンその他の用途にピックスマップを供給できます。テキストと同様、一部のピックスマップは特定の言語環境に固有のものです。それらのピックスマップはローカライズされていなければなりません。

デスクトップはピックスマップおよびイメージのキャッシュを管理します。XmGetPixmapByDepth() 関数は、要求されたピックスマップを探してそれらのキャッシュを検索します。要求されたピックスマップがピックスマップのキャッシュに存在せず、対応するイメージがイメージのキャッシュにない場合

は、XmGetPixmapByDepth() 関数が、要求されたイメージ名と名前が一致する X ビットマップ・ファイルを検索します。XmGetPixmapByDepth() 関数は、ファイルを検索するために XtResolvePathname() 関数を呼び出します。要求されたイメージ名が絶対パス名の場合、そのパス名が XtResolvePathname() 関数の検索パスです。そうでない場合、XmGetPixmapByDepth() 関数は次のように検索パスを構築します。

- XBMLANGPATH 環境変数が設定されている場合は、その変数の値が検索パスです。
- XBMLANGPATH は設定されていないけれども XAPPLRESDIR が設定されている場合、XmGetPixmapByDepth() 関数は、エントリに $XAPPLRESDIR、ユーザのホーム・ディレクトリ、ベンダ依存のシステム・ディレクトリを含むデフォルト検索パスを使用します。
- XBMLANGPATH および XAPPLRESDIR のいずれも設定されていない場合は、XmGetPixmapByDepth() 関数は、エントリにユーザのホーム・ディレクトリとベンダ依存のシステム・ディレクトリを含むデフォルト検索パスを使用します。

これらのパスには *%B* 置換フィールドがあるかもしれません。XtResolvePathname() 関数への呼び出しのたびに、XmGetPixmapByDepth() 関数は *%B* を要求されたイメージ名に置き換えます。パスには、XtResolvePathname() 関数が受け取るその他の置換フィールドがあるかもしれません。特に、XtResolvePathname() 関数は *%L* を表示の言語文字列に置き換え、*%l*、*%t*、*%c* を表示の言語文字列のコンポーネント (ベンダ依存) に置き換えます。置換フィールド *%T* は常にビットマップにマップされ、*%S* は常にヌルにマップされます。

デフォルトでは文字列からピックスマップへのコンバータがないので、ピックスマップは通常 XmGetPixmap() への呼び出しで最初にピックスマップを取り出すことにより、アプリケーションによって作成時に設定されます。XmGetPixmap() は、現在のロケールを使用してピックスマップをどこから検索するか決定します (ピックスマップの検索とロケールの関係については、XmGetPixmap() のマニュアル・ページを参照してください)。

フォント・リソース

表 2–9 はローカライズできるフォント・リソースの一覧です。すべての XmFontList リソースの型は XmFontList です。ほぼすべての場合、フォント・リスト要素を指定するときにはフォント・セットが使用されます。唯一の例外

は、ユーザの文字セットに現れない文字データを表示するときです (たとえば数学の記号や読者の注意を引く記号を表示する場合)。

表 2–9 ローカライズできるフォント・リソース

ウィジェット・クラス	リソース名
VendorShell	*buttonFontList:
VendorShell	*defaultFontList:
VendorShell	*labelFontList:
VendorShell	*textFontList:
XmBulletinBoard	*buttonFontList:
XmBulletinBoard	*defaultFontList:
XmBulletinBoard	*labelFontList:
XmBulletinBoard	*textFontList:
XmLabel[Gadget]	*fontList:
XmList	*fontList:
XmMenuShell	*buttonFontList:
XmMenuShell	*defaultFontList:
XmMenuShell	*labelFontList:
XmText	*fontList:
XmTextField	*fontList:

オペレーティング・システム国際化対応関数

表 2–10 は、共通オープン・ソフトウェア環境の基本オペレーティング・システムの国際化対応関数の一覧です。

ロケールでは 1 文字のコード化に 1〜4 バイト必要であるという仮定の下に、アプリケーションが適切なロケール管理を実行するべきです。

表 2–10 基本オペレーティング・システム国際化対応関数

ロケール管理	シングルバイト	マルチバイト	ワイド文字
mb <-> wcwc 間の変換		mbtowc	wctomb
		mbstowcs	wcstombs
分類	isalpha		isalpha
	is*		isw*
			wctype
ケース・マッピング	tolower		towlower
	toupper		towupper
形式に関する雑多な情報		localeconv	
		nl_langinfo	
数値の形式		strtol	wcstol
		strtod	wcstod
			wcstoi
時刻／通貨の形式		strftime	wcsftime
		strptime	
		strfmon	

表 2–10 基本オペレーティング・システム国際化対応関数 続く

ロケール管理	シングルバイト	マルチバイト	ワイド文字
文字列のコピー		strcat	wcscat
		strcpy	wcsncat
		strncat	wcscpy
		strncpy	wcsncpy
文字列の照合		strcoll	wcscoll
			wcsxfrm
文字列の操作	strlen	mblen	wcscmp
			wcsncmp
文字列の検索	strchr		wcschr
	strcspn		wcscspn
	strpbrk		wcspbrk
	strrchr		wcsrchr
	strspn		wcsspn
	strtok		wcstok
			wcswcs
			wcscspn
I/O 表示幅			wcwidth [1]
			wcswidth

表 2–10 基本オペレーティング・システム国際化対応関数 続く

ロケール管理	シングルバイト	マルチバイト	ワイド文字
I/O 出力		printf	printf
		vprintf	vprintf
		sprintf	sprintf
		vsprint	vsprint
		fprintf	frpintf
		vfprint	vfprint
I/O スキャン		scanf	scanf
		sscanf	sscanf
		fscanf	fscanf
I/O 文字	getc		fgetwc
	gets		fgetws
	putc		fputwc
	puts		fputws
	ungetc		ungetwc
メッセージ		gettxt	
		catopen	
		catgets	
		catclose	
コードセット変換		iconv_open	
		iconv	
		iconv_close	

1. これらの関数は、端末を使用するアプリケーションに提供されます。グラフィカル・ユーザ・インタフェース (GUI) アプリケーションには、これらの関数を使用しないでください。その代わりに、31ページの「シンプル・テキスト」にリストされているフォント・メトリック関数を使用してスペーシングを決定します。

第 3 章

国際化対応と分散ネットワーク

この章では、国際化対応と分散ネットワークに関連するタスクについて説明します。

変換の概念

この節では、8 ビットのユーザ名と 8 ビット・データが、ftp、メール、デスクトップ・クライアント間のクライアント間通信などの通信ユーティリティによりネットワーク上で通信できる方法を説明します。

データを通信するにあたって、まず考慮すべき点が 3 つあります。

- 送信側のコード・セットと受信側のコード・セット
- 通信プロトコルが 8 ビット・データを許可するか、または 7 ビット・コード・データに限られているか (たとえば、日本のインターネットは JIS (日本工業規格) コード・データを 7 ビット・プロトコルで通信します。)
- プロトコル規則ごとにある変換エンコーディングの型。実際に必要となる変換は、使用される個々のプロトコルに依存します。

リモート・ホストがローカル・ホストと同じコード・セットを使用する場合は、次の事項が真になります。

- プロトコルが 8 ビット・データを使用できる場合は、変換は必要ありません。
- プロトコルが 7 ビット・データしか使用できない場合は、8 ビット・コード・ポイントを 7 ビット ASCII 値にマップする必要があります。これは、iconv() フレームワークと、次の 7 ビット・エンコード方法の 1 つを使って達成できます。
  - 8 ビット・データを、POSIX.2 仕様の uuencode および uudecode アルゴリズムに指定されているとおりにマップする。
  - 任意で、8 ビット・データをプロトコルで定義されているように 7 ビット変換エンコーディングにマップする。たとえば、Xlib の 7 ビット ISO2022 や MIME (Multipurpose Intrenet Message Extensions: 多目的インターネット・メッセージ拡張機能) の base64 があります。

リモート・ホストのコード・セットがローカル・ホストのコード・セットと異なるときは、次の 2 つの場合が当てはまります。必要な変換は、使用される特定のプロトコルに依存します。

- プロトコルが 8 ビット・データを使用できる場合、プロトコルはどちら側が iconv() 変換を行うかを指定し、また回線上でのエンコーディングを指定する必要があります。プロトコルによっては、可能なコード・セットのすべてをエンコードでき、文字レパートリーを識別する機能のある 8 ビット変換エンコーディングを推奨します。
- プロトコルが 7 ビット・データしか使用できない場合は、7 ビット変換エンコーディングと文字レパートリーの識別が必要です。

iconv インタフェース

ネットワーク環境では、通信し合っているシステムのコード・セットと通信のプロトコルによって、ユーザの指定したデータが意味ある方法でリモート・システムに送信されるように、データの変換方法が決定されます。(ユーザ名でなく) ユーザ・データを送信側のコード・セットから受信側のコード・セットに変換したり、プロトコルに準拠するよう 8 ビット・データを 7 ビット形式に変換する必要があります。このことを達成するには一様なインタフェースが必要です。

次の例では、iconv_open()、iconv()、iconv_close() の使い方を説明し、iconv() インタフェースの使用方法を示しています。この変換を実行するには、 iconv_open() の次に必ず iconv() を続けてください。7 ビット変換および 8 ビット変換という用語は、それぞれ 7 ビット・データと 8 ビット・データの変換エンコーディングの意味で使用します。

送信側と受信側が同じコード・セットを使用している場合

- プロトコルが 8 ビット・データを使用できる場合は、同じコード・セットが使用されているので 8 ビット・データを使用します。変換は必要ありません。
- プロトコルが 7 ビット・データしか使用できない場合は、iconv() を使用します。
  - 送信側

    ```
    cd =iconv_open(locale_codeset, uuencoded );
    ```

  - 受信側

    ```
    cd = iconv_open(uucode, locale_codeset );
    ```

送信側と受信側が異なるコード・セットを使用している場合

- プロトコルが 8 ビット・データを使用できる場合
  - 送信側

    ```
    cd = iconv_open(locale_codeset, 8-bitinterchange );
    ```

  - 受信側

    ```
    cd = iconv_open(8-bitinterchange, locale_codeset );
    ```

- プロトコルが 7 ビット・データしか使用できない場合は、次のようにします。
  - 送信側

    ```
    cd = iconv_open(locale_codeset, 7-bitinterchange );
    ```

  - 受信側

    ```
    cd = iconv_open(7-bitinterchange, locale_codeset );
    ```

locale_codeset は、そのロケールのアプリケーションによって使用されているコード・セットです。nl_langinfo()(CODESET) 関数を使用して現在のロケールに関連付けられたコード・セットを獲得できますが、それは変換名が nl_langinfo()(CODESET) 関数からの戻り値と一致するかどうかは実装に依存します。

表 3–1 に、さまざまな条件のもとで変換を実行する際の iconv() の使用方法を示します。プロトコルによっては他の変換が必要な場合もあります。

表 3–1　変換を実行するための iconv の使用方法

	同じコード・セットを使用するシステムとの通信 (例: XYZ)		異なるコード・セットを使用するシステムとの通信、または受信側のコード・セットが不明	
使用する変換	7 ビット・プロトコル	8 ビット・プロトコル	7 ビット・プロトコル	8 ビット・プロトコル
コード XYZ	無効	最適	無効	リモート・コード・セットが不明の場合無効
7 ビット変換 ISO2022	OK	OK	最適	OK
8ビット変換 ISO2022, ISO10646	無効 [1]	OK	無効	最適
7 ビットタグなし引用符付き印刷可能な uucode	OK	OK	コード・セットの識別が必要	コード・セットの識別が必要
8 ビットタグなし base64	無効	OK	コード・セットの識別が必要	コード・セットの識別が必要

1. 無効とは、選択したコード・セットとプロトコル型には変換エンコーディングは使用すべきでないという意味です。

状態を持つ変換と状態を持たない変換

コード・セットは、状態を持つ (ステートフルな) エンコーディングと状態を持たない (ステートレスな) エンコーディングの 2 つのカテゴリに分類できます。

状態を持つエンコーディング

状態を持つエンコーディングは、特定のコード値に関連付けられた文字セットを変換するのに、シフトイン／シフトアウトなどの制御コードのシーケンスを使用します。

たとえば、コンパウンド・テキストでは、文字データの流れの中で日本語 16 ビット・データの開始を示すのにコントロール・シーケンス「ESC$(B」を使用できます。また、「ESC(B」は、そのダブルバイト文字データの終了と 8 ビット ASCII データの開始を示すのに使用できます。状態を持つエンコーディングでは、ビット値 0x43 はシフト状態が不明の場合解釈できません。EBCDIC アジア・コード・セットは、シフトイン制御とシフトアウト制御を、それぞれダブルバイトとシングルバイト・エンコーディング間の入れ換えに使用します。

別のコード・セットへの状態を持つエンコーディング変換を行うために記述されるコンバータは、特別な処理が必要なためにやや複雑になります。

状態を持たないエンコーディング

状態を持たないコード・セットは、次の 2 つのうちの 1 つに分類できます。

- シングルバイト・コード・セット (ISO8859 ファミリなど)
- マルチバイト・コード・セット (日本語用 PC コード、通称 Shift-JIS (SJIS) など)

マルチバイト・コード・セットという用語は、1 つの文字をエンコードするのに 1 つ以上のバイトを必要とするコード・セットにも使います。マルチバイト・コード・セットは状態を持たないと見なされます。

注 - コード・セットが同じ文字セットを表すときに限り、変換してください。

シンプル・テキストの基本的なデータ交換

あるプログラムがリモート・ホストにある別のプログラムにデータを通信するとき、元のマシンのコード・セットから受信側のコード・セットへデータを変換する必要が生じることがあります。たとえば、PC コードを使用している PC システムが、ISO/EUC (国際標準化機構／拡張 UNIX コード) のエンコーディングを使用しているワークステーションと通信する必要があるときなどです。また、プログラムがあるコード・セットのデータを獲得したが、そのデータを別のコード・セットで表示しなければならない場合も同様です。そのような変換をサポートするために、XPG4 `iconv()` 関数定義に基づく標準プログラム・インタフェースが用意されています。

コード・セット変換を行うすべてのコンポーネントは、変換のインタフェースとして `iconv()` 関数を使用してください。システムは、変換のデフォルト・セットをカスタマイズする機構の他に、広範囲の変換を提供することを期待されます。

iconv 変換関数

1 つのコード・セットから別のコード・セットへ変換する共通の方法は、テーブルを使う方法です。場合によりテーブルが大きすぎることもあります。この時は、アルゴリズムによる方法が望ましい方法です。多様な要求事項を満たすために、XPG4 にコード・セット変換のフレームワークが定義されています。そのフレームワークでは、あるコード・セットから別のコード・セットに変換するにはコンバータを開き、変換を実行し、コンバータを閉じます。`iconv()` 関数には `iconv_open()`、`iconv()`、`iconv_close()` があります。

コード・セット・コンバータは、関数 `iconv_open()`、`iconv()`、`iconv_close()` のフレームワークの下にあります。これらの関数により、数種類の異なる型のコンバータを提供し、使用することができます。アプリケーションは、あるコード・セットの文字を別のコード・セットの文字に変換するのにこれらの関数を呼び出します。`iconv()` フレームワークにより、コンバータが一様に提供できるようになります。このようなコンバータのアクセスおよび使用は、X/Open XPG4 で標準化されています。

X クライアント間 (ICCCM) 変換関数

Xlib は変換するために次の関数を提供します。

X ICCCM マルチバイト関数	ICCCM ワイド文字関数
`XmbTextPropertyToTextList()`	`XwcTextPropertyToTextList()`
`XmbTextListToTextProperty()`	`XwcTextListToTextProperty()`

注 - `libXm()` ライブラリは、`XmStringConvertToCT()` 関数と `XmStringConvertFromCT()` 関数を提供します。しかし、特定の XmString タグにはハードコードされた前提条件があるため、それらの関数は推奨できません。たとえば、タグが `bold()` の場合、`XmStringConvertToCT()` は処理系に依存します。さまざまなプラットフォームで、この関数の動作を世界のすべての地域では保証できません。

詳細は、127ページの「ローカライズされたテキストのクライアント間通信規約」を参照してください。

ウィンドウ・タイトル

タイトルを設定する一般的な方法は、リソースを使用することです。しかしアプリケーションが直接ウィンドウのタイトルを設定する場合は、ローカライズされたタイトルをウィンドウ・マネージャに送信しなければなりません。次のガイドラインの他に、XICCEncodingStyle() に定義された XCompundTextStyle() エンコーディングを使用してください。

- コンパウンド・テキストは、XmbTextListToTextProperty() か XwcTextListToTextProperty() のいずれかで作成できます。
- ローカライズされたタイトルは、WMShell() ウィジェットの XmNtitle() リソースと XmNtitleEncoding() リソースを使用して表示できます。ローカライズされたアイコン名は、TopLevelShell() ウィジェットの XmNiconName() リソースか XmNiconNameEncoding() リソースを使用して表示できます。
- ダイアログ・ボックスのローカライズされたタイトルは、XmBulletinBoard() ウィジェットの XmNdialogTitle() リソースを使用して表示できます。
- ウィンドウ・マネージャは、ローカライズされた文字列を表示するのに適切なフォント・リストを持っていなければなりません。

次の例は、ローカライズされたタイトルとアイコン名を表示します。この例ではコンパウンド・ストリングからコンパウンド・テキストが作成されます。

```
#include       <nl_types.h>
Widget         toplevel;
Arg            al[10];
int            ac;
XTextProperty  title;
char           *localized_string;
nl_catd        fd;

XtSetLanguageProc( NULL, NULL, NULL );
fd = catopen( "my_prog", 0 );
localized_string = catgets(fd, set_num, mes_num, "defaulttitle");
XmbTextListToTextProperty( XtDisplay(toplevel), &localized_string,
      1, XCompoundTextStyle, &title);
ac = 0;
XtSetArg(al[ac], XmNtitle, title.value); ac++;
XtSetArg(al[ac], XmNtitleEncoding, title.encoding); ac++;
XtSetValues(toplevel, al, ac);
```

ウィジェットでなくウィンドウを使用している場合は、XmbSetWMProperties() 関数がローカライズされた文字列を適切な XICCEncodingStyle に自動的に変換します。

メールの基本的な交換

一般に、電子メール (email) の手順は、受信側のロケールに関する情報をあらかじめ知ってメッセージをそれに最適化する方法ではなく、正規ラベル付け形式です。つまり電子メールの世界では、受信側が異なるロケールにいるかもしれないことを常に想定しておくべきです。デスクトップの世界では、デフォルトの電子メール転送は SMTP (簡易メール転送プロトコル) です。SMTP は 7 ビット転送チャネルだけをサポートします。

この他に、電子メールの手順については次のような点があげられます。

- 送信側プログラムは、(ユーザが別の手順を指定しなければ) デフォルトで、本文の部分を送信側の転送チャネルの標準形式に変換し、使用される文字エンコーディングで本体の部分にラベル付けします。
- 受信側プログラムは文字エンコーディングをサポートできるかどうか知るために本文の部分を見ます。サポートできる場合はローカル文字セットに変換します。

さらに、メッセージには MIME 形式が使用されるため、8 ビットから 7 ビットへの変換は、組み込み MIME 転送エンコーディングを使用して実行されます (base64 または引用符付き表示可能形式)。RFC (Request for Comments) 1521 MIME 標準仕様を参照してください。

エンコーディングとコード・セット

コード・セットを理解するには、まず文字セットを理解することが必要です。文字セットは、文字を表すのに使用するエンコーディング値は考慮せずに、1 つ以上の言語の特定のニーズに基づいてあらかじめ定義された文字の集まりです。どのコード・セットを使用するかの選択は、ユーザのデータ処理要件に依存します。特定の文字セットは、異なるエンコーディング・スキーマを使用してエンコードされます。たとえば、ASCII 文字セットは英語の中で見つけられる文字のセットを定義します。JIS (日本工業規格) の文字セットは、日本語で使用される文字のセットを定義します。英語および日本語の文字セットは、両方とも異なるコード・セットを使用してエンコードされます。

ISO2022 標準は、コード化文字セットを、文字セットと、各文字とそのビット・パターンの一対一の関係を定義する細かい規則の集まりとして定義します。コード・セットは、システムが文字を識別するのに使用するビット・パターンを定義します。

コード・ページはコード・セットに似ていますが、コード・ページの仕様は 16 列 × 16 行のマトリクスに基づくという制限があります。各列と行の交わりが、コード化文字を定義します。

コード・セット

共通オープン・ソフトウェア環境のコード・セット・サポートは、ISO (国際標準化機構) と、ユーザのデータ処理ニーズを満たす業界標準のコード・セットを提供する業界標準コード・セットに基づいています。

システム上の各ロケールは、どのコード・セットを使用するか、またそのコード・セット内の文字がどのように処理されるかを定義します。システムには複数のロケールをインストールできるので、複数のコード・セットがシステム上の異なるユーザによって使用されます。システムが複数の異なるコード・セットを使用する複数のロケールで構成される一方、すべてのシステム・ユーティリティはシステムが単一のコード・セットの下で動作していると想定します。

ほとんどのコマンドは、ロケールが使用している下位のコード・セットについては何も知りません。コード・セットの知識は、コード・セットに依存しないライブラリ・サブルーチン (国際化対応ライブラリ) によって隠されています。コード・セットに依存しないライブラリ・サブルーチンは、コード・セット依存サブルーチンに情報を渡します。

多くのプログラムが ASCII に依存しているため、すべてのコード・セットには 7 ビット ASCII コード・セットが適正なサブセットとして含まれています。7 ビット ASCII コード・セットはサポートされているすべてのコード・セットに共通なので、7 ビット ASCII コード・セットの文字はポータブル文字セットと呼ばれることがあります。

7 ビット ASCII コード・セットは ISO646 定義に基づいており、制御文字、句読文字、数字 (0-9)、大文字と小文字の英字のアルファベットが含まれます。

コード・セットの構造

各コード・セットは 2 つの主な領域に分かれます。

- グラフィック・レフト (GL) 0-7 列
- グラフィック・ライト (GR) 8-F 列

各コード・セットの最初の 2 列は、制御文字用に ISO 標準が確保しています。C0 と C1 は、それぞれグラフィック・レフトとグラフィック・ライトの領域用の制御文字を表すのに使用されます。

注 - PC コード・セットは、グラフィック文字をエンコードするのに C1 コントロール領域を使用します。

残りの 6 列はグラフィック文字をエンコードするのに使用されます (図 3–1 参照)。グラフィック文字は印刷可能な文字と見なされます。制御文字は、デバイスとアプリケーションによって特別な機能を表すために使用されます。

図 3–1　コード・セットの概要

制御文字

ISO 定義に基づき、制御文字は動作を開始、変更、停止します。制御文字はグラフィック文字ではありませんが、場合によってはグラフィック表現を持つことができます。ISO646-IRV 文字セットの制御文字は、サポートされているすべてのコード・セット内にあります。C0 制御文字のエンコードされた値は、あらゆるコード・セットを通じて一貫しています。

グラフィック文字

各コード・セットは、各文字に固有のコード値が与えられるように、1 つ以上の文字セットに分かれると考えられます。ISO 標準は 6 つの列を文字のエンコーディングのために確保しており、グラフィック文字を制御文字の列でエンコードできません。

シングルバイト・コード・セット

1 バイトの 8 ビットすべてを使用するコード・セットは、ヨーロッパ、中東その他のアルファベット言語をサポートします。そのようなコード・セットはシングルバイト・コード・セットと呼ばれます。シングルバイト・コード・セットは、文字のエンコーディングを制御文字を除いて 191 文字までに制限しています。

マルチバイト・コード・セット

マルチバイト・コード・セットは、特定の文字をエンコードするのに必要なバイト数に関わらず、すべての可能なコード・セットを指します。オペレーティング・システムは 1 文字をエンコードするのに何ビットでもサポートできるので、マルチバイト・コード・セットには、8 ビット、16 ビット、32 ビット、あるいはそれ以上のビットでエンコードされた文字を含めることが可能です。シングルバイト・コード・セットもマルチバイト・コード・セットと考えられます。

EUC (拡張 UNIX コード) コード・セット

EUC コード・セットは、一部の文字セットの中で文字を識別するのに制御文字を使用します。エンコーディング規則は ISO2022 の 7 ビット・データと 8 ビット・データのエンコーディングの定義に基づいています。EUC コード・セットは一部の文字セットを区切るのに制御文字を使用します。

EUC という用語はそのような一般的なエンコーディング規則を指します。EUC に基づくコード・セットは EUC エンコーディング規則に準拠しますが、特定の場合に関連付けられた特定の文字セットも識別します。たとえば、日本語用 eucJP は EUC エンコーディング規則による JIS 文字のエンコーディングを指します。

最初のセット (CS0) には常に ISO646 文字セットが含まれます。他のすべてのセットは、最上位ビット (MSB) を 1 に設定しなければならず、文字をエンコードするのに何バイトでも使用できます。さらに、1 つのセットの中のすべての文字は次のようでなければなりません。

- すべての文字をエンコードするのに同じバイト数を使用する

- 列表示幅 (固定幅の端末での列数) が同じである

3 番目のセット (CS2) の各文字の前には、常に制御文字 SS2 (シングルシフト 2、0x8e) が付きます。EUC に準拠するコード・セットは、3 番目のセットを識別する目的以外では制御文字 SS2 を使用しません。

4 番目のセット (CS3) の各文字の前には、常に制御文字 SS3 (シングルシフト 3、0x8f) が付きます。EUC に準拠するコード・セットは、4 番目のセットを識別する目的以外では制御文字 SS3 を使用しません。

ISO EUC コード・セット

次のコード・セットは、ISO (国際標準化機構) により設定された定義に基づいています。

- ISO646-IRV
- ISO8859-1
- ISO8859-x
- eucJP
- eucTW
- eucKR

ISO646-IRV

ISO646-IRV コード・セットは、7 ビット・エンコーディングに基づく情報処理に使用されるコード・セットを定義します。このコード・セットに関連付けられた文字セットは ASCII 文字から得られます。

ISO8859-1

ISO8859-1 エンコーディングは、その他の ISO、ANSI (米国規格協会)、ECMA (欧州コンピュータ製造者協会) のコード拡張技術に基づき、それらと互換性のあるシングルバイトのエンコーディングです。ISO8859 エンコーディングは、各メンバが独自の文字セットを持つコード・セットのファミリを定義します。7 ビット ASCII コード・セットは、ISO8859 ファミリの各コード・セットの適切なサブセットです。

ISO8859-1 コード・セットは ISO Latin-1 コード・セットと呼ばれ、2 つの文字セットから成ります。

- ISO646-IRV グラフィック・レフト、7 ビット ASCII 文字セット
- ISO8859-1 グラフィック・ライト (ラテン) 文字セット

これらを組み合わせた文字セットには、デンマーク語、オランダ語、英語、フィンランド語、フランス語、ドイツ語、アイスランド語、イタリア語、ノルウェー語、ポルトガル語、スペイン語、スウェーデン語などの西欧諸語に必要な文字が含まれます。

ASCII コード・セットが英語のアルファベット順に順序を定義する一方、GR (グラフィック・ライト) 文字は特定のどの言語によっても順序付けされません。言語固有の順序はロケールによって定義されます。

その他の ISO8859 コード・セット

この節ではその他の重要な ISO8859 コード・セットをリストします。各コード・セットには ASCII 文字とそのコード・セット独自の文字があります。

ISO8859-2

ラテン・アルファベット、No.2、東欧

- アルバニア語
- チェコスロヴァキア語
- 英語
- ドイツ語
- ハンガリー語
- ポーランド語
- ルーマニア語
- セルビア-クロアチア語
- スロヴァキア語
- スロヴェニア語

ISO8859-5

ラテン／キリル・アルファベット

- ブルガリア語

- 白ロシア (ベロルシア) 語
- 英語
- マケドニア語
- ロシア語
- ウクライナ語

ISO8859-6

ラテン／アラビア語アルファベット

- 英語
- アラビア語

ISO8859-7

ラテン／ギリシャ語アルファベット

- 英語
- ギリシャ語

ISO8859-8

ラテン／ヘブライ語アルファベット

- 英語
- ヘブライ語

ISO8859-9

ラテン／トルコ語アルファベット

- デンマーク語
- オランダ語
- 英語
- フィンランド語
- フランス語

- ドイツ語
- アイルランド語
- イタリア語
- ノルウェー語
- ポルトガル語
- スペイン語
- スウェーデン語
- トルコ語

eucJP

日本語用 EUC はシングルバイト文字とマルチバイト文字 (2 バイトと 3 バイト) から成ります。エンコーディングは ISO2022 に準拠し、JIS および EUC の定義に基づきます。

表 3–2 eucJP のエンコーディング

CS	エンコーディング		文字セット
cs0	0xxxxxxx		ASCII
cs1	1xxxxxxx	1xxxxxxx	JIS X0208-1990
cs2	0x8E	1xxxxxxx	JIS X0201-1976
cs3	0x8F	1xxxxxxx 1xxxxxxx	JIS X0212-1990

JIS X0208-1990

情報交換用の日本語のグラフィック文字セットのコード (1990 年版) です。この中には特殊文字が 147、数字が 10、ひらがな文字が 83、カタカナ文字が 86、ラテン文字が 52、ギリシャ文字が 48、キリル文字が 66、線描画要素が 32、漢字が 6355 含まれます。

JIS X0201

カタカナを 63 文字含む、情報変換用コードです。

JIS X0212-1990

情報変換用の日本語のグラフィック文字セットの補助コード (1990 年版) です。この中には、追加の特殊文字が 21、追加のギリシャ文字が 21、追加のキリル文字が 26、追加のラテン文字が 27、発音区別符号の付いたラテン文字が 171、追加の漢字が 5801 含まれます。

eucTW

繁体字用 EUC はシングルバイト文字とマルチバイト文字 (2 バイトと 4 バイト) を含む文字から成るエンコーディングです。EUC エンコーディングは ISO2022 に準拠し、中華民国と EUC によって定義されたとおり、CNS (Chinese National Standard) に基づきます。表 3–3 を参照してください。

表 3–3 eucTW のエンコーディング

CS	エンコーディング			文字セット
cs0	0xxxxxxx			ASCII
cs1	1xxxxxxx	1xxxxxxx		CNS 11643.1992 - plane 1
cs2	0x8EA2	1xxxxxxx	1xxxxxxx	CNS 11643.1992 - plane 2
cs3	0x8EA3	1xxxxxxx	1xxxxxxx	CNS 11643.1992 - plane 3
	0x8EB0	1xxxxxxx	1xxxxxxx	CNS 11643.1992 - Plane 16

CNS 11643-1992 は、中国標準変換コード用に 16 の面を定義します。各面は、8836 文字 (94 × 94) までサポートできます。現在は、面 1～7 のみ文字が割り当てられています。表 3–4 は、CNS 11643-1992 標準の 16 の各面を示しています。

表 **3–4** CNS 11643-1992 標準の 16 面

面	定義	文字数	**EUC** エンコーディング
1	最も多く使用される	6085	A1A1-FDCB
2	2 番目に多く使用される	7650	8EA2 A1A1 - 8EA2 F2C4
3	Exec. Yuen EDP [1]センター	6148	8EA3 A1A1 - 8EA3 E2C6
4	RIS [2]、ベンダ定義	7298	8EA4 A1A1 - 8EA4 EEDC
5	MOE はほとんど使用しない[3]	8603	8EA5 A1A1 - 8EA5 FCD1
6	MOE による変形文字セット 1	6388	8EA6 A1A1 - 8EA6 E4FA
7	MOE による変形文字セット 2	6539	8EA7 A1A1 - 8EA7 E6D5
8	未定義	0	8EA8 A1A1 - 8EA8 FEFE
9	未定義	0	8EA9 A1A1 - 8EA9 FEFE
10	未定義	0	8EAA A1A1 - 8EAA FEFE
11	未定義	0	8EAB A1A1 - 8EAB FEFE
12	ユーザ定義文字 (UDC)	0	8EAC A1A1 - 8EAC FEFE
13	UDC	0	8EAD A1A1 - 9EAD FEFE
14	UDC	0	8EAE A1A1 - 8EAE FEFE
15	UDC	0	8EAF A1A1 - 8EAF FEFE
16	UDC	0	8EB0 A1A1 - 8EB0 FEFE

1. EDP: 予算、会計、統計の中央理事会
2. RIS: 居住地情報システム
3. MOE: 文部省

eucKR

韓国語用 EUC は、シングルバイト文字とマルチバイト文字から成るエンコーディングです (表 3–5 参照)。エンコーディングは ISO2022 に準拠し、KSC (韓国語標準コード) セットと EUC 定義に基づきます。

表 3–5 eucKR のエンコーディング

CS	エンコーディング		文字セット
cs0	0xxxxxxx		ASCII
cs1	1xxxxxxx	1xxxxxxx	KS C 5601-1992
cs2			Not used
cs3			Not used

KSC 5601-1992 (1992 年度版情報変換用韓国語文字セットのコード) には、特殊文字が 432、アラビア数字およびローマ数字が 30、ハングル・アルファベットが 94、ローマ文字が 52、ギリシャ文字が 48、ラテン文字が 27、日本語の文字が 169、ロシア文字が 66、線描画要素が 68、あらかじめ作成されたハングルが 2344、ハンジャが 4888 含まれます。

1 つのハングル文字は子音と母音から成ります。ハングルのほとんどの単語はハンジャの単語でも表現できます。ハンジャは繁体字のセットであり、現在韓国語圏の人々に使用されています。各ハンジャには意味があるので、ほとんどの場合ハングルよりも明確です。

第 4 章

Motif 依存性

この章では、Motif における国際化対応に関連する作業について説明します。

ロケール管理

言語環境という用語は、アプリケーションがユーザの指定したロケールで正しく実行するために必要な、ローカライズされたデータのセットを言います。言語環境は、特定の言語に関連する規則を提供します。また、言語環境は、外部に格納されたすべてのデータから成ります。このデータは、アプリケーションが使用するローカライズされた文字列やテキストなどです。たとえば、アプリケーションが表示するメニュー項目は、アプリケーションがサポートする言語別に、別ファイルに格納されている可能性があります。この型のデータは、リソース・ファイル、UID (ユーザ・インタフェース定義) ファイル、(XPG3 準拠システムの) メッセージ・カタログに格納できます。

1 つの言語環境は、1 つのアプリケーションが動作するときに確立されます。アプリケーションが実行される言語環境は、アプリケーション・ユーザによって、環境変数 (POSIX システムの `LANG` または `LC_*`) か xnlLanguage リソースのいずれか

により設定されます。その後アプリケーションはユーザの指定に基づいて言語環境を設定します。アプリケーションはこの設定を、XtSetLanguageProc() 関数に確立された言語プロシージャの setlocale() 関数を使って行うこともできます。これにより Xt は、XtResolvePathname() 関数がリソース、ビットマップ、UIL (ユーザ・インタフェース言語) ファイルを見つけるのに使用するディスプレイごとの指定言語文字列をキャッシュします。

言語プロシージャを提供するアプリケーションは、独自のプロシージャを提供するか、あるいは Xt デフォルト・プロシージャを使用することができます。どちらの場合も、アプリケーションは、ツールキットを初期化する前、また (XtAppInitialize() 関数を呼び出すなどして) リソース・データベースを読み込む前に XtSetLanguageProc() を呼び出すことにより、言語プロシージャを確立します。言語プロシージャがインストールされると、Xt は初期リソース・データベースの構築中にその言語プロシージャを呼び出します。Xt は言語プロシージャが返す値をディスプレイごとの指定言語文字列に使用します。

デフォルトの言語プロシージャは次のタスクを実行します。

- 次のようにしてロケールを設定します。

  ```
  setlocale(LC_ALL, language);
  ```

  language は xnlLanguage リソースの値です。または、xnlLanguage リソースが設定されていない場合は空の文字列 (“”) です。xnlLanguage リソースが設定されていない場合、一般的にロケールは環境変数 (POSIX システムの LANG) から得られます。

- 設定されたロケールがサポートされているかを確認するために、XSupportsLocale()関数を呼び出します。サポートされていない場合、警告メッセージが発行されロケールは C に設定されます。

- 空の文字列を指定する XSetLocaleModifiers() 関数を呼び出します。

- 現在のロケールの値を返します。ANSI C に基づくシステムでは、その値は次の呼び出しの結果です。

  ```
  setlocale(LC_ALL, NULL);
  ```

アプリケーションは、次のように XtSetLanguageProc() 関数を呼び出すことにより、デフォルト言語プロシージャを使用できます。

```
XtSetLanguageProc(NULL, NULL, NULL);
    .
    .
toplevel = XtAppInitialize(...);
```

デフォルトでは、Xt は言語プロシージャを何もインストールしません。アプリケーションが XtSetLanguageProc() 関数を呼び出さないと、Xt はディスプレイごとの指定言語文字列に xnlLanguage リソースが設定されている場合は、その値を使用します。xnlLanguage リソースが設定されていない場合は、Xt は言語文字列を LANG 環境変数から獲得します。

注 - このプロセスから得られるディスプレイごとの指定言語文字列は実装に依存します。また、いったん言語文字列が確立されると、Xt は言語文字列を調べる公共の手段を提供しません。

独自の言語プロシージャを提供することにより、アプリケーションは言語文字列を設定する際にどんなプロシージャでも使用できます。

フォント管理

デスクトップは、テキストを表示するためにフォント・リストを使用します。フォントは、指定された文字セットの文字を表現するグリフのセットを定義します。フォント・セットは、指定されたロケールまたは言語のテキストを表示するのに必要なフォントの集まりです。フォント・リストは、使用されるフォントか、フォント・セットか、その両方のリストです。Motif にはフォント・リストを作成する簡易関数があります。

フォント・リスト構造

デスクトップには、テキスト表示用にフォント・リストが必要です。フォント・リストは、フォント構造体か、フォント・セットか、その両方のリストであり、それぞれに識別のためのタグがあります。フォント・セットは、現在の言語のすべての文字が確実に表示できるようにします。フォント構造体により、すべての文字が確実に表示できるようにするのは、(ロケールのコード・セットからグリフ索引への変換を含めて) プログラマの責任です。

フォント・リストの各エントリの形式は、{*tag, element* } の組み合わせです。*element* は単一のフォントかフォント・セットのいずれかです。アプリケーションは、単一のフォントとフォント・セットのどちらからでもフォント・リスト・エントリを作成できます。たとえば、次のコード・セグメントはフォント・セットのフォント・リスト・エントリを作成します。

```
char font1[] =
        "-adobe-courier-medium-r-normal--10-100-75-75-M-60";
font_list_entry = XmFontListEntryLoad (displayID, font1,
        XmFONT_IS_FONTSET, "font_tag");
```

XmFontListEntryLoad() 関数は、フォントを読み込むか、またはフォント・セットを作成して読み込みます。この関数の 4 つの引き数は次のとおりです。

displayID	フォント・リストが使用されるディスプレイ
fontname	フォント名かベース・フォント名リストのいずれか (*nametype* 引き数による) を表す文字列
nametype	*fontname* 引き数がフォント名とベース・フォント名リストのどちらを表すかを指定する値
tag	そのフォント・リスト・エントリのタグを表す文字列

nametype 引き数が XmFONT_IS_FONTSET の場合、XmFontListEntryLoad() 関数は *fontname* 引き数の値からフォント・セットを現在のロケールに作成します。フォント・セットに指定されたフォントの文字セットは、ロケールに依存します。*nametype* が XmFONT_IS_FONT の場合、XmFontListEntryLoad() 関数は *fontname* にあるフォントをオープンします。どちらの場合も、フォントまたはフォント・セットがフォント・リスト・エントリに置かれます。

次のコード例は、新しいフォント・リストを作成し、そこにエントリfont_list_entry を追加します。

```
XmFontList font_list;
XmFontListEntry font_list_entry;
    .
    .
font_list = XmFontListAppendEntry (NULL, font_list_entry);
XmFontListEntryFree (font_list_entry);
```

フォント・リストが作成されると、XmFontListAppendEntry() 関数がそこに新しいエントリを追加します。次の例は、XmFontListEntryCreate() 関数を使用して既存のフォント・リストに新しいフォント・リスト・エントリを作成します。

```
XFontSet font2;
char *font_tag;
XmFontListEntry font_list_entry2;
    .
    .
font_list_entry2 = XmFontListEntryCreate (font_tag,
     XmFONT_IS_FONTSET, (XtPointer)font2);
```

font2 パラメータは XCreateFontSet() 関数に返された XFontSet を指定します。XmFontListEntryCreate() 関数の引き数は font_tag、XmFONT_IS_FONTSET、font2 であり、それぞれタグ、型、フォントです。タグおよびフォント・セットは、フォント・リスト・エントリの {*tag, element*} の組み合わせです。

このエントリをフォント・リストに追加するには、XmFontListAppendEntry() 関数を再度使用します。このときのみ最初のパラメータが既存のフォント・リストを指定します。

```
font_list = XmFontListAppendEntry(font_list, font_list_entry2);
XmFontListEntryFree(font_list_entry2);
```

フォント・リストの例

リソース・ファイルにフォント・リストを指定する形式は、リストに含まれるものがフォントか、フォント・セットか、その両方かによって異なります。

フォントの獲得

フォントを獲得するには、フォントとオプションのフォント・リスト要素タグを指定します。

- タグがある場合は、その前に = (等号記号) を付けます。
- タグがない場合は、= (等号記号) を使用しないでください。

複数のフォントを指定するエントリは、, (カンマ) で区切られます。

フォント・セットの獲得

フォント・セットを獲得するには、ベース・フォント・リストとオプションのフォント・リスト要素タグを指定します。

- タグがある場合は、その前に = (等号記号) ではなく : (コロン) を付けます。
- タグがない場合でも、コロンは必ず付けなければなりません。コロンはリソース宣言においてフォントをフォント・セットと区別するからです。

ベース・フォント・リストに指定された複数のフォントは ; (セミコロン) で区切られます。複数のフォント・セットを指定するエントリは , (カンマ) で区切られます。

フォント・リスト要素タグがない場合のフォントの指定

フォント・リスト要素タグがない場合、デフォルトの XmFONTLIST_DEFAULT_TAG が使用されます。次に例をいくつか示します。

- デフォルトのフォント・リスト要素タグを使用してフォントを指定します。

```
*fontList: fixed
 *fontList: \
 -adobe-courier-medium-r-normal--10-100-75-75-M-60-iso8859-1
```

- フォント・リスト要素タグを指定します。

```
*fontList: fixed=ROMAN, 8x13bold=BOLD
```

- デフォルトのフォント・リスト要素タグと明示タグで 2 つのフォントを指定します。

```
*fontList: fixed, 8x13bold=BOLD
```

フォント・リスト要素タグがない場合のフォント・セットの指定

フォント・リスト要素タグがない場合、デフォルトの XmFONTLIST_DEFAULT_TAG が使用されます。次にフォント・セット指定の例をいくつか示します。

- フォント・リスト要素タグを指定せずに、Xlib にフォントを選択させます。

```
*fontList: -dt-application-medium-r-normal-*-m*-*-*-*-m-*
```

- Xlib にフォントを選択させ、フォント・リスト要素タグに *MY_TAG* を指定させます。

```
*fontList: -dt-application-medium-r-normal-*-m*-*-*-*-m*:MY_TAG
```

- Xlib にフォントを選択させ、フォント・リスト要素タグにボールド指定させ、その他に関してはデフォルトのフォント・リスト要素タグを使用させます。

```
*fontList: -dt-application-medium-r-normal-*-m*-*-*-*-m-*:,\
   -dt-application-medium-r-normal-style2-m*-*-*-*-m-*:BOLD
```

フォント・リスト形式

XmFontList() データ型は次の要素に関連付けられた 1 つ以上のエントリを含むことができます。

XFontStruct	フォントの charset でエンコードされたテキスト (フォント・エンコード・テキスト) を描画するのに使用できる X フォント

XFontSet	ロケールでエンコードされたテキスト (ローカライズされたテキスト) を描画するのに使用できる XFontStruct フォントの集まり

次の形式は、文字列から XmFontList へのコンバータによって使用されます。

```
XmFontList    := <fontentry> {', 'fontentry}

fontentry     := <fontname><fontid>
              | <baselist><fontsetid>

baselist      := <fontname>{';'<fontname>}

fontsetid     := ':'<string> | <defaultfontset>

fontname      := <XLFD string>

fontid        := '='<string> | <defaultfont>

XLFD string   := XLFD 仕様を参照
defaultfont   := NULL
defaultfontset:= ':'NULL
string        := 改行をのぞく ISO646IRV のすべての文字
```

XmFontList が指定された fontentry は、フォントかフォント・セットのいずれかを指定できます。どちらの場合も、コンパウンド・ストリング (XmString) 内のセグメントにより ID (fontid または fontsetid) が参照できます。

defaultfont と defaultfontset はいずれもデフォルトの fontentry を定義できますが、XmFontList 単位にデフォルトは 1 つだけしかありません。

XmFONTLIST_DEFAULT_TAG 識別子は、XmString が描画されるときに、常にデフォルトの fontentry を参照します。デフォルトの fontentry が指定されない場合は、最初の fontentry が描画に使用されます。

リソース・コンバータは、作成されたすべてのフォント・セットが同じロケールに関連付けられるように、単一のロケールで動作します。

注 - 実装によっては、ロケールのコード・セット名を、特別な charset ID (fontid または fontsetid) として XmFontList 文字列の中に確保します。このためアプリケーション開発者は、プラットフォーム間でポータブルなアプリケーションを作成したい場合は、コード・セット名を使用しないように注意してください。

ローカライズされたテキストの描画

コンパウンド・ストリングは、プログラムを変更しなくても数多くのフォントでテキストを表示できるようにテキストをエンコーディングする手段です。デスクトップは、Text ウィジェットと TextField ウィジェット以外のすべてのテキストを表示するのにコンパウンド・ストリングを使用します。この節は、コンパウンド・ストリングの構造、コンパウンド・ストリングとフォント・リスト間の対話を説明し(コンパウンド・ストリングの表示方法を決定する)、国際化対応プロセスにとって重要な点にフォーカスをあてます。

コンパウンド・ストリングのコンポーネント

コンパウンド・ストリングは、タグ長値のセグメントから成る内部のエンコーディングです。意味上では、コンパウンド・ストリングには、表示されるテキスト、フォント・リストの要素に一致するタグ (フォント・リスト要素タグ)、表示する方向を示すインジケータなどのコンポーネントがあります。

コンパウンド・ストリングのコンポーネントは、次の 4 つのいずれかの型になります。

- フォント・リスト要素タグ
  - フォント・リスト要素タグ XmFONTLIST_DEFAULT_TAG は、テキストが現在のロケールのコード・セットでエンコードされることを示します。
  - その他のフォント・リスト要素タグは、あとでテキストをフォント・リストの特定のエントリに一致させるために使用されます。
- 方向識別子
- 文字列のテキスト。国際化対応アプリケーションでは、テキストは大きく 2 種類に分けられます。ローカライズされた処理が必要なテキストと、そうでないテキストです。
- セパレータ

コンパウンド・ストリングの各コンポーネントを説明します。

フォント・リスト要素タグ　　コンパウンド・ストリングのテキスト・コンポーネントをフォント・リストのフォントまた

	はフォント・セットに相関させる文字列の値を示します。
方向	文字がキーボードに入力される順序と、その文字が画面に表示される順序の関係を示します。たとえば、英語、フランス語、ドイツ語、イタリア語の表示順序は左から右であり、ヘブライ語とアラビア語の表示順序は右から左です。
テキスト	表示されるテキストを示します。
セパレータ	値がない特殊な形式のコンパウンド・ストリングのコンポーネントを示します。セパレータは他のセグメントを区切るのに使用されます。

共通デスクトップ環境では、コンパウンド・ストリングを表示するために、テキスト・コンポーネントで識別される指定されたフォント・リスト要素タグを使用します。指定されたフォント・リスト要素タグは、新しいフォント・リスト要素タグが見つかるまで使用します。共通デスクトップ環境は、現在のコード・セットに対して正しいフォントに一致する特別なフォント・リスト要素タグ XmFONTLIST_DEFAULT_TAG を提供します。XmFONTLIST_DEFAULT_TAG はフォント・リストのデフォルト・エントリを識別します。詳細は、87ページの「コンパウンド・ストリングとフォント・リスト」を参照してください。

コンパウンド・ストリングの方向セグメントは、テキストが表示される方向を指定します。方向は左から右または右から左です。

コンパウンド・ストリングとリソース

コンパウンド・ストリングは、Text ウィジェットと TextField ウィジェット以外のすべてのテキストを表示するのに使用されます。コンパウンド・ストリングは、表示できるように適切なウィジェット・リソースに設定されます。たとえば、PushButton ウィジェットのラベルは Label ウィジェットから引き継がれ、リソースは XmNlabelString で型は XmString です。このことは、リソースがコンパウンド・ストリングの値を予想していることを意味します。コンパウンド・ストリングはプログラムで作成したり、リソース・ファイルで定義することが可能です。

プログラムでのコンパウンド・ストリングの設定

アプリケーションは、XmStringCreateLocalized() コンパウンド・ストリング簡易関数を使用してコンパウンド・ストリングを作成することで、このリソースをプログラムで設定できます。

この関数は、現在のロケールのエンコーディングでコンパウンド・ストリングを作成し、自動的に XmFONTLIST_DEFAULT_TAG をフォント・リスト・エントリ・タグに設定します。

次のコードの一部分は、プログラムを使用してプッシュ・ボタンに XmNlabelString リソースを設定するための 1 つの方法を示します。

```
#include        <nl_types.h>
Widget          button;
Args            args[10];
int             n;
XmString button_label;
nl_msg my_catd;
(void)XtSetLanguageProc(NULL,NULL,NULL);
    .
    .
button_label = XmStringCreateLocalized (catgets(my_catd, 1, 1,
                                         "default label"),
                                         XmFONTLIST_DEFAULT_TAG);

/* Create an argument list for the button */
n = 0;
XtSetArg (args[n], XmNlabelString, button_label); n++;

/* Create and manage the button */
button = XmCreatePushButton (toplevel, "button'', args, n);
XtManageChild (button);
XmStringFree (button_label);
```

コンパウンド・ストリングのデフォルト・ファイルへの設定

国際化対応プログラムでは、ボタン・ラベル用のラベル文字列は外部リソースから獲得してください。たとえば、ボタン・ラベルはプログラムではなくリソース・ファイルから得られます。次の例では、プッシュ・ボタンは form1 と呼ばれる Form ウィジェットの子であると想定します。

```
*form1.button.labelString: Push Here
```

ここで、デスクトップの文字列からコンパウンド・ストリングへのコンバータは、リソース・ファイル・テキストからコンパウンド・ストリングを生成します。このコンバータは常に XmFONTLIST_DEFAULT_TAG を使用します。

コンパウンド・ストリングとフォント・リスト

デスクトップがコンパウンド・ストリングを表示する場合、デスクトップはセグメントのフォント・リスト要素タグを使用して、各セグメントをフォントまたはフォント・セットに関連付けます。アプリケーションは希望するフォントまたはフォント・セットを読み込んで、そのフォントまたはフォント・セットと、それに関連付けられたフォント・リスト要素タグを含むフォント・リストを作成し、同じタグでコンパウンド・ストリング・セグメントを作成していなければなりません。

デスクトップは、コンパウンド・ストリングをフォント・リスト・エントリに割り当てるとき、次のように設定された検索プロシージャに従います。

1. デスクトップは、フォント・リストで、コンパウンド・ストリングに指定されたフォント・リスト要素タグと厳密に一致するものを検索します。一致点が見つかると、コンパウンド・ストリングはそのフォント・リスト・エントリに割り当てられます。

2. コンパウンド・ストリングとフォント・リスト間の一致点が見つからない場合は、デスクトップは、コンパウンド・ストリングをフォント・リスト要素タグに関係なくフォント・リストの最初の要素に割り当てます。

後方互換を保つため、厳密な一致点が見つからない場合は、コンパウンド・ストリングまたはフォント・リストの XmFONTLIST_DEFAULT_TAG の値が、XmSTRING_DEFAULT_CHARSET のタグでコンパウンド・ストリングまたはフォント・リストのエントリを作成した結果のタグに一致します。

図 4–1 は、フォント・リスト要素タグが XmFONTLIST_DEFAULT_TAG 以外に設定される場合の、コンパウンド・ストリング、フォント・セット、フォント・リスト間の関係を示しています。

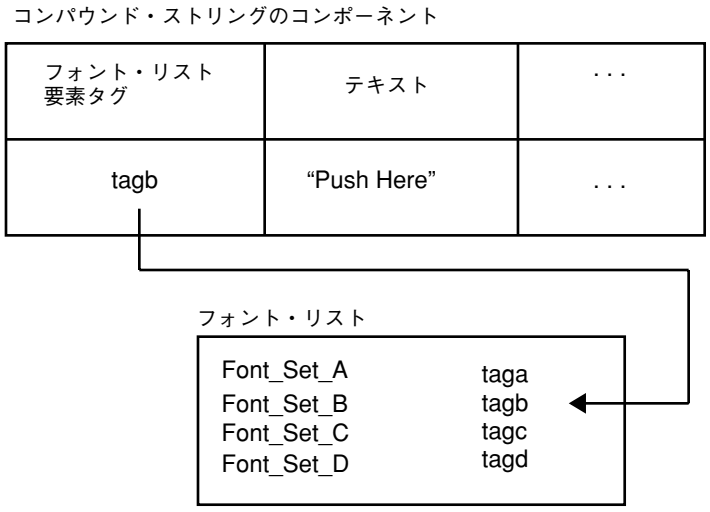

図 4–1　フォント・リスト要素タグが XmFONTLIST_DEFAULT_TAG でない場合の、コンパウンド・ストリング、フォント・セット、フォント・リスト間の関係

次の例は、tagb と呼ばれるタグの使用方法を示しています。

```
XFontSet                *font1;
XmFontListEntry         font_list_entry;
XmFontList              font_list;
XmString                label_text;
char                    **missing;
int                     missing_cnt;
char                    *del_string;
char                    *tagb;          /* Font list element tag */
char                    *fontx;         /* Initialize to XLFD or font alias */
char                    *button_label;  /* Contains button label text */
                .
                .
font1 = XCreateFontSet (XtDisplay(toplevel), fontx, & missing,
                        & missing_cnt, & def_string);
font_list_entry = XmFontListEntryCreate (tagb, XmFONT_IS_FONTSET,
                        (XtPointer)font1);
font_list = XmFontListAppendEntry (NULL, font_list_entry);
XmFontListEntryFree (font_list_entry);
label_text = XmStringCreate (button_label, tagb);
```

XCreateFontSet() 関数はフォント・セットを読み込み、XmFontListEntryCreate() 関数はフォント・リスト・エントリを作成します。アプリケーションは、エントリを作成してそれを既存のフォント・リストに追加するか、または新しいフォント・リストを作成しなければなりません。どちらの場合も XmFontListAppendEntry() 関数を使用します。適切な場所にフォント・

リストがないので、前述のコード例ではフォント・リスト引き数が NULL の値になっています。XmFontListAppendEntry() 関数は、単一のエントリ font_list_entry を持つ font_list という新しいフォント・リストを作成します。font_list に新しいエントリを追加するには、同じプロシージャに従い、ヌルでないフォント・リスト引き数を提供してください。

図 4–2 は、フォント・リスト要素タグが XmFONTLIST_DEFAULT_TAG に設定される場合の、コンパウンド・ストリング、フォント・セット、フォント・リストの関係を示しています。この場合、値のフィールドはロケール・テキストです。

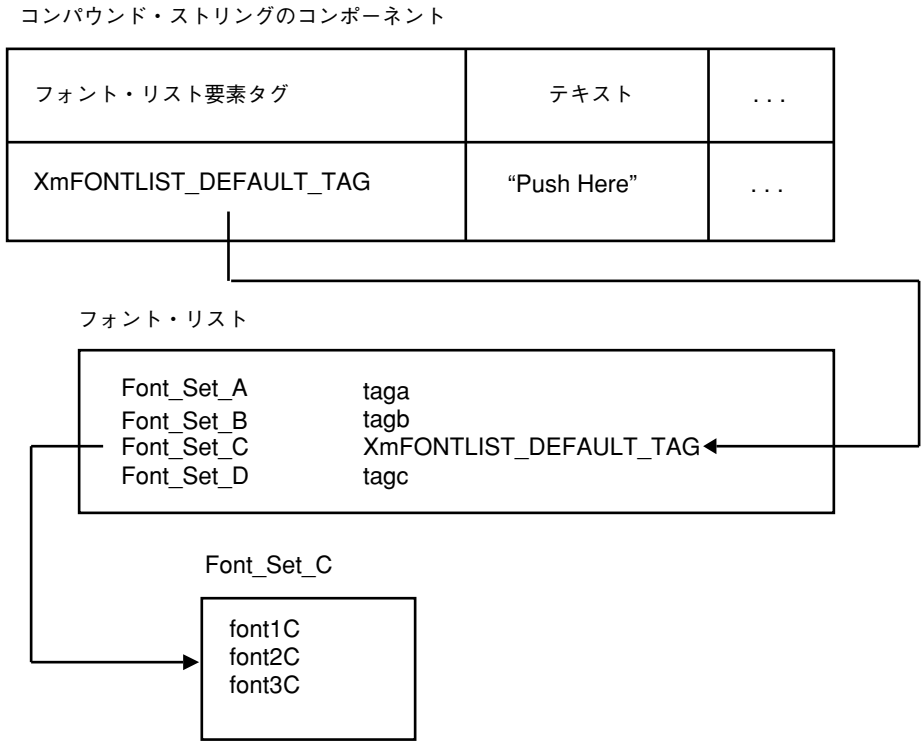

図 4–2　フォント・リスト要素タグが XmFONTLIST_DEFAULT_TAG に設定されている場合の、コンパウンド・ストリング、フォント・セット、フォント・リスト間の関係

ここで、デフォルト・タグは Font_Set_C を指しています。Font_Set_C は、その言語で文字を表示するのに必要なフォントを識別します。

Text ウィジェットおよび TextField ウィジェットとフォント・リスト

Text ウィジェットと TextField ウィジェットは、テキスト情報を表示します。そのためには、ウィジェットは、情報を表示するのに使う正しいフォントを選択できなければなりません。Text ウィジェットと TextField ウィジェットは、正しいフォントを見つけるために、次のように設定された検索パターンに従います。

1. ウィジェットは、フォント・リストでフォント・リスト要素タグ XmFONTLIST_DEFAULT_TAG を持つフォント・セットであるエントリを検索します。一致点が見つかった場合は、そのフォント・リスト・エントリを使用します。それ以上の検索は行われません。

2. 一致点が見つからなければ、ウィジェットは、フォント・リストでフォント・セットを指定するエントリを検索します。ウィジェットは最初に見つかったフォント・セットを使用します。

3. フォント・セットが見つからない場合は、ウィジェットはフォント・リストの 1 番目のフォントを使用します。

フォント・セットを使用すると、ロケール内のどの文字に対してもグリフが確実に存在します。

ローカライズされたテキストの入力

システム環境では、VendorShell ウィジェット・クラスは入力メソッドへのインタフェースを提供するために拡張されます。VendorShell クラスは、ジオメトリ管理で 1 つの子ウィジェットだけをコントロールする一方、入力メソッドへのインタフェースに必要なすべてのコンポーネントが管理できるように、VendorShell クラスに拡張が追加されます。これらのコンポーネントにはステータス領域、プリエディット領域、メイン・ウィンドウ領域があります。

入力メソッドが、ステータス領域かプリエディット領域あるいはその両方を必要とするとき、VendorShell ウィジェットは自動的にステータス領域とプリエディット領域のインスタンスを生成し、それらのジオメトリ・レイアウトを管理します。ステータス領域もプリエディット領域もすべて VendorShell ウィジェットにより内部で管理され、クライアントからはアクセスできません。VendorShell ウィ

ジェットの子としてインスタンスを生成されたウィジェットは、メイン・ウィンドウ領域と呼ばれます。

VendorShell ウィジェットが使用する入力メソッドは XmNinputMethod リソース (たとえば @im=alt) により決定されます。デフォルト値ヌルは、VendorShell が作成されたときに、ロケールに関連付けられたデフォルトの入力メソッドを選択することを示します。このように、ユーザはロケールを設定するか、XmNinputMethod リソースを設定するか、あるいは両方設定することにより、どの入力メソッドが選択されるかに影響を与えることができます。入力メソッド名を決定するために、ロケール名は XmNinputMethod リソースと連結されます。ロケール名はこのリソースに指定してはなりません。XmNinputMethod リソースのモディファイアの名前を @im=*modifier* 形式の中に指定しなければなりません。*modifier* は、どの入力メソッドが選択されているかを修飾するのに使う文字列です。

VendorShell ウィジェットは、入力メソッドを共用する複数のウィジェットをサポートできます。しかし、どんな時でもキーボードのフォーカスを持てる (たとえば、キー・プレスのイベントを受信してそれを入力メソッドに送信する) ウィジェットは 1 つだけです。複数のウィジェット (Text ウィジェットなど) をサポートするには、それらのウィジェットが VendorShell ウィジェットの子孫でなければなりません。

注 - VendorShell ウィジェット・クラスは、TransientShell および TopLevelShell ウィジェット・クラスのスーパークラスです。このように、TopLevelShell または Diagshell のインスタンスの生成は、本質的に VendorShell ウィジェット・クラスのインスタンスの生成です。

VendorShell ウィジェットは、子孫の 1 つが XmText[Field] インスタンスである場合のみ入力マネージャとして動作します。XmText[Field] インスタンスが VendorShell ウィジェットの子孫として作成されるとすぐに、VendorShell は現在のロケールによって指示される特定の入力メソッドに必要な領域を作成します。XmText[Field] インスタンスがマップされずにただ作成された場合でも、VendorShell は前述のようなジオメトリ管理の動作を行います。

VendorShell ウィジェットは次のことを行います。

- システムにインストールされたロケールがサポートしているマルチバイト文字の入力および出力を、アプリケーションが処理できるようにします。
- 入力メソッドのインスタンスを、XmIm 参照関数に定義されたとおりに管理します。

- OffTheSpot、OverTheSpot、Root、None のいずれかのモードで、プリエディット領域でのプリエディットをサポートします。ローカライズされたテキストは、フォーカスを変更することにより、複数の Text 子ウィジェット・ツリー内のどの Text 子ウィジェットにも入力できます。
- 子孫ウィジェットのジオメトリ管理を提供します。

ジオメトリ管理

VendorShell ウィジェットは、入力メソッドのユーザ・インタフェース・コンポーネントのジオメトリ管理とフォーカス管理を必要に応じて提供します。ロケールがそれを保証する場合 (たとえばロケールが日本語の EUC (拡張 UNIX コード) ロケールの場合)、VendorShell ウィジェットは、必要なプリエディット領域かステータス領域あるいはその両方のジオメトリを自動的に割り当てて管理します。

現在行われているプリエディットによっては補助領域が必要になります。補助領域が必要な場合は、VendorShell ウィジェットは補助領域の手続きを生成して管理します。通常、VendorShell ウィジェットの子は複数の Text および TextField ウィジェットを管理できるコンテナ・ウィジェット (XmBulletinBoard ウィジェットまたは XmRowColumn ウィジェットなど) であり、ユーザからのマルチバイト文字の入力ができます。このシナリオでは、すべての Text ウィジェットは同一の入力メソッドを共用します。

注 - ステータス領域、プリエディット領域、補助領域にはアプリケーションのプログラマはアクセスできません。たとえば、アプリケーションのプログラマがステータス領域のウィンドウ ID にアクセスすることは想定されていません。それらのコンポーネントは必要に応じて VendorShell ウィジェット・クラスが管理するので、ユーザはそれらのコンポーネントのインスタンスの生成や管理について考える必要はありません。

アプリケーションのプログラマは、VendorShell ウィジェット・クラスの XmNpreedetType リソースを介して、入力メソッドのユーザ・インタフェース・コンポーネントの動作をいくらかコントロールできます。OffTheSpot モードと OverTheSpot モードについては、13ページの「入力メソッド」を参照してください。

ジオメトリ管理は、すべての入力メソッドのユーザ・インタフェースのコンポーネントに及びます。アプリケーション・プログラム・ウィンドウ (TopLevelShell ウィジェット) がサイズ変更されると、入力メソッドのユーザ・インタフェースのコンポーネントもそれに応じてサイズ変更され、その中のプリエディットされた文字

列は必要に応じて再配置されます。もちろん、シェル・ウィンドウのサイズ変更ポリシーが true であることを想定しています。

VendorShell ウィジェットが作成されるとき、特定の入力メソッドがステータス領域、プリエディット領域、あるいはその両方を必要とする場合、VendorShell のサイズはそれらのコンポーネントが必要とする領域を考慮します。プリエディット領域とステータス領域が必要とする特別な領域は、VendorShell ウィジェットの領域の一部です。それらの領域も、サイズ変更する必要がある場合は VendorShell ウィジェットに管理されます。

それらの領域 (ステータス領域およびプリエディット領域) の潜在的な手続きの作成のために、現在使用されている入力メソッドによって、VendorShell ウィジェット領域のサイズは必ずしも子のサイズにぴったり合うように伸縮する必要はありません。VendorShell ウィジェット領域のサイズは、子のジオメトリとこれらの入力メソッドのユーザ・インタフェース領域のジオメトリの両方が入るように伸縮します。VendorShell ウィジェットと子ウィジェット (メイン・ウィンドウ領域) 間の高さには差があるかもしれません (たとえば 20 ピクセル)。幅のジオメトリは、入力メソッドのユーザ・インタフェースのコンポーネントに影響されません。

まとめると、子に要求されたサイズは可能であれば受け付けられます。VendorShell の実際のサイズは子よりも大きい場合もあります。

VendorShell ウィジェットと子のジオメトリを指定する要求は、互いに矛盾しない限り、または VendorShell ウィジェットのサイズ変更能力の制約内であれば行われます。矛盾する場合は、子のウィジェット・ジオメトリ要求が優先します。たとえば、子ウィジェットのサイズが 100 × 100 に指定された場合、VendorShell のサイズも 100 × 100 に指定されます。子ウィジェットのサイズが 100 × 100 になるのに対して、VendorShell のサイズは結果的に 100 × 120 になります。子ウィジェットのサイズが指定されない場合、独自のサイズ指定を使用する必要があれば VendorShell は子ウィジェットを縮小します。たとえば、VendorShell のサイズが 100 × 100 に指定され、子のサイズは指定されない場合、子ウィジェットのサイズは 100 × 80 になります。VendorShell ウィジェットがサイズ変更を禁止されている場合は、子のジオメトリ要求がどうであっても VendorShell ウィジェットは独自のジオメトリ指定を使用します。

フォーカス管理

多数の文字を使用する言語 (日本語や中国語など) には、ユーザがその言語で対話的に文字を構成できる入力メソッドが必要です。このような言語には、端末のキーボードに適正にマップできる数をはるかに超える文字が存在するからです。

そのような言語で文字を構成する対話的なプロセスをプリエディットといいます。プリエディット自体は入力メソッドによって処理されます。しかし、プリエディットのユーザ・インタフェースはシステム環境により決定されます。入力メソッドとシステム環境の間にインタフェースが存在する必要があります。これは、システム環境の VendorShell ウィジェットを介して実行できます。

図 4–3 は日本語のプリエディットの例を示しています。反転表示された文字列がプリエディット中の文字列です。この文字列は、特定のウィンドウへのフォーカスを与えることにより、異なるウィンドウに移動できます。しかし、プリエディット・セッションは一度に 1 つだけです。

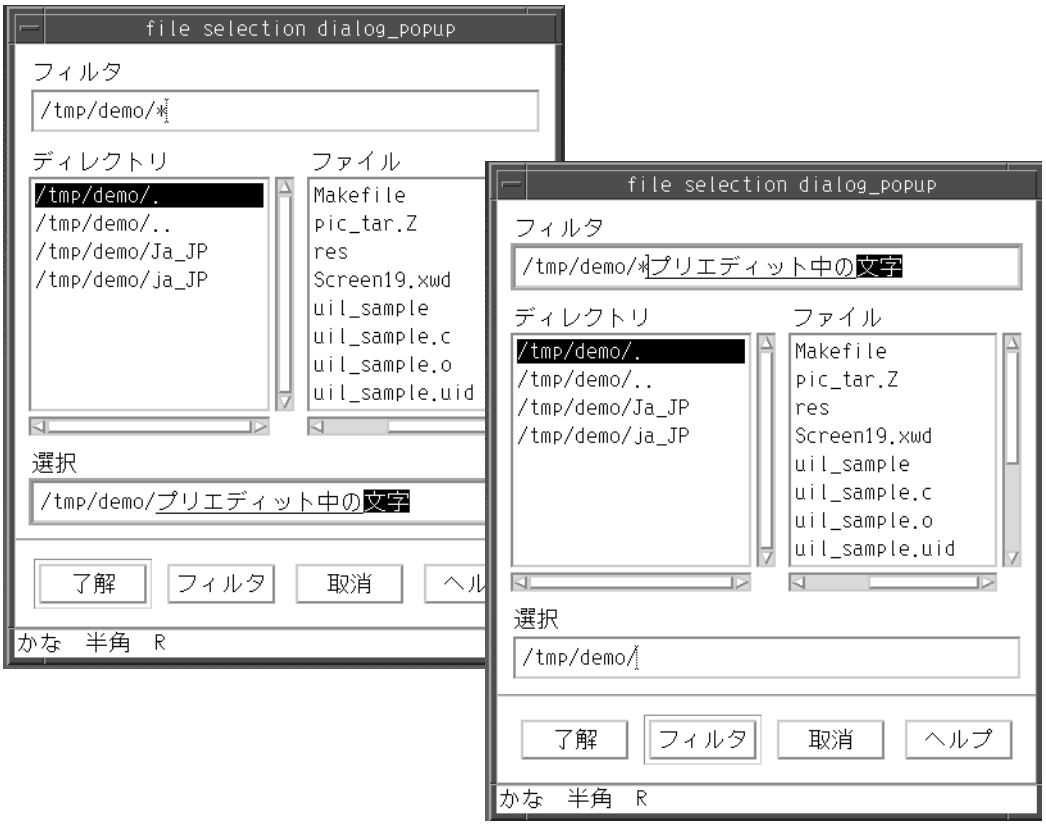

図 4–3　日本語のプリエディットの例

フォーカス管理の例として、TopLevelShell ウィジェット (VendorShell ウィジェットのサブクラス) が、5 つの XmText ウィジェットを子に持つ XmBulletinBoard ウィジェットの子 (メイン・ウィンドウ領域) を持っていると想定します。そのロケールにはプリエディット領域が必要で、OverTheSpot モードが

指定されていると想定します。VendorShell ウィジェットは入力メソッドの手続きを 1 つだけ管理するので、プリエディット領域は TopLevelShell ウィジェットの内部で一度に 1 つだけしか実行できません。フォーカスがある Text ウィジェットから他の Text ウィジェットへ移動される場合は、処理中の現在のプリエディット文字列も、現在フォーカスを持っている Text ウィジェットのトップに移動されます。以前の Text ウィジェットへのキー処理は一時的に中断されます。以降の入力メソッドのインタフェース (プリエディット完了時の文字列の送信等) は、フォーカスされた新しい Text ウィジェットに対して行われます。

プリエディットされている文字列は、マウスをクリックするなどの動作でフォーカスの位置に移動できます。

エンド・ユーザがプリエディットを終えてすでに確定した文字列は再変換できません。いったん文字列が構成されると、それは確定されます。文字列を確定するというのは、その文字列がプリエディット領域からクライアントのフォーカス・ポイントへ移動することです。

国際化対応ユーザ・インタフェース言語

マルチバイト文字の文字列を文字列リテラルとして解析する機能が UIL (ユーザ・インタフェース言語) に追加されました。UIL ファイルの作成は、目的の言語の特徴を使って UID (ユーザ・インタフェース定義) ファイルを記述することで実行できます。

国際化対応ユーザ・インタフェース言語のプログラミング

UIL コンパイラは、標準にはない charset をロケール・テキストとして解析します。そのためには、UIL コンパイラがどのロケール・テキストとも同じロケールで実行する必要があります。

ウィジェットのロケール・テキストにフォント・セット (複数のフォント) が必要な場合、フォント・セットがリソース・ファイル内に指定されなければなりません。*font* パラメータはフォント・セットをサポートしません。

UIL に特定の言語を使用するために、UIL ファイルが目的の言語の特徴に応じて記述され、UID ファイルにコンパイルされます。ローカライズされたテキストを格納

している UIL ファイルは、それを実行するロケールにコンパイルする必要があります。

文字列リテラル

文字列リテラルの例を次に示します。cur_charset の値は常に default_charset の値に設定されており、それによって文字列リテラルがロケール・テキストを格納できます。

default_charset の値でロケール・テキストを文字列リテラルに設定するには、次のように入力します。

```
XmNlabelString = 'XXXXXX';
```

または、次のように入力します。

```
XmNlabelString = #default_charset``XXXXXX'';
```

ロケール・テキストのエンコーディングに一致する LANG 環境変数で UIL ファイルをコンパイルしてください。そうでない場合、文字列リテラルは正しくコンパイルされません。

フォント・セット

フォント・セットは UIL ソース・プログラミングを介して設定することはできません。フォント・セットが必要な場合は、次の例のように必ずリソース・ファイルにフォント・セットを設定してください。

```
*fontList: -*-r-*-20-*:
```

フォント・リスト

UIL はフォント・リストの作成に使用する 3 つの関数 (FONT 、FONTSET 、FONT_TABLE) を持っています。FONT 関数と FONTSET 関数はフォント・リスト・エントリを作成します。FONT_TABLE 関数はそれらのフォント・リスト・エントリからフォント・リストを作成します。

FONT 関数はフォント指定を含むフォント・リスト・エントリを作成します。引き数は XLFD フォント名を表す文字列です。FONTSET 関数は、フォント・セット指定を含むフォント・リスト・エントリを作成します。引き数は、ベース名フォント・リストを表すカンマで区切られた XLFD フォント名のリストです。

FONT と FONTSET には両方とも、フォント・リスト・エントリのフォント・リスト要素タグを指定するオプションの CHARACTER_SET 宣言パラメータがあります。どちらの場合も、CHARACTER_SET 宣言パラメータが指定されない場合は、UIL が次のようにフォント・リスト要素タグを決定します。

- モジュールに CHARACTER_SET 宣言がなく、uil コマンドが –s オプションと共に呼び出されたか、あるいは Uil() 関数が use_setlocale_flag セットで開始された場合、フォント・リスト要素タグは XmFONTLIST_DEFAULT_TAG です。
- それ以外の場合、UIL コンパイル環境にフォント・リスト要素タグが設定されている場合は、フォント・リスト要素タグは LANG 環境変数のコード・セット・コンポーネントです。LANG 環境変数が設定されていない、またはコード・セットがない場合は、フォント・リスト要素タグは XmFALLBACK_CHARSET の値です。

FONT_TABLE 関数は、FONT または FONTSET により作成された、カンマで区切られたフォント・リスト・エントリのリストからフォント・リストを作成します。その結果のフォント・リストは、フォント・リスト・リソースの値として使用できます。そのようなリソースの値として単一のフォント・リスト・エントリが提供される場合は、UIL はそのエントリをフォント・リストに変換します。

リソース・ファイルの作成

必要であれば、次の例のように入力メソッド関連リソースをリソース・ファイルに設定してください。

```
*preeditType:    OverTheSpot、OffTheSpot、Root、または None
```

環境の設定

ロケールを区別するアプリケーションの場合は、UID ファイルを適切なディレクトリに設定してください。UIDPATH または XAPPLRESDIR 環境変数を適切な値に設定してください。

たとえば、英語環境で uil_sample プログラムを実行するには (LANG 環境変数は en_US) 、$HOME/en_US ディレクトリにラテン文字で uil_sample.uid を設定するか、またはあるディレクトリに uil_sample.uid を設定して UIDPATH 環境変数に uil_sample.uid ファイルの完全パス名を設定してください。

uil_sample プログラムを日本語環境で実行するには (LANG 環境変数は ja_JP)、$HOME/ja_JP ディレクトリに日本語の (マルチバイト) 文字で uil_sample.uid ファイルを作成するか、または uil_sample.uid を一意のディレクトリに配置して UIDPATH 環境変数に uil_sample.uid ファイルの完全パス名を設定してください。次のリストは可能な変数を指定します。

%U	UID ファイル文字列を指定します。
%N	アプリケーションのクラス名を指定します。
%L	xnlLanguage リソースか LC_CTYPE カテゴリの値を指定します。
%l	xnlLanguage リソースか LC_CTYPE カテゴリの言語コンポーネントを指定します。

XAPPLRESDIR 環境変数が設定されている場合、MrmOpenHierarchy() 関数が次の順番で UID ファイルを検索します。

1. UID ファイル・パス名
2. $UIDPATH
3. *%U*
4. $XAPPLRESDIR/*%L*/uid/*%N*/*%U*
5. $XAPPLRESDIR/*%l*/uid/*%N*/*%U*
6. $XAPPLRESDIR/uid/*%N*/*%U*
7. $XAPPLRESDIR/*%L*/uid/*%U*
8. $XAPPLRESDIR/*%l*/uid/*%U*
9. $XAPPLRESDIR/uid/*%U*
10. $HOME/uid/*%U*
11. $HOME/*%U*
12. /usr/lib/X11/*%L*/uid/*%N*/*%U*
13. /usr/lib/X11/*%l*/uid/*%N*/*%U*
14. /usr/lib/X11/uid/*%N*/*%U*
15. /usr/lib/X11/*%L*/uid/*%U*
16. /usr/lib/X11/*%l*/uid/*%U*
17. /usr/lib/X11/uid/*%U*
18. /usr/include/X11/uid/*%U*

XAPPLRESDIR 環境変数が設定されていない場合は、MrmOpenHierarchy() 関数は XAPPLRESDIR 環境変数の代わりに $HOME を使用します。

UIL の default_charset 文字セット

default_charset 文字列リテラルについては、どんな文字でも有効な文字列リテラルとして設定できます。たとえば、LANG 環境変数が el_GR の場合は、default_charset の文字列リテラルにはギリシャ文字が入ります。LANG 環境変数が ja_JP の場合は、default_charset の文字列リテラルには日本語の EUC でエンコードされたすべての日本語の文字が入ります。

文字列リテラルに文字セットが設定されていない場合、文字列リテラルの文字セットは cur_charset に設定されます。また、システム環境では、cur_charset の値は常に default_charset に設定されています。

例: uil_sample

図 4–4 は、英語と日本語の環境での UIL のプログラム例を示します。

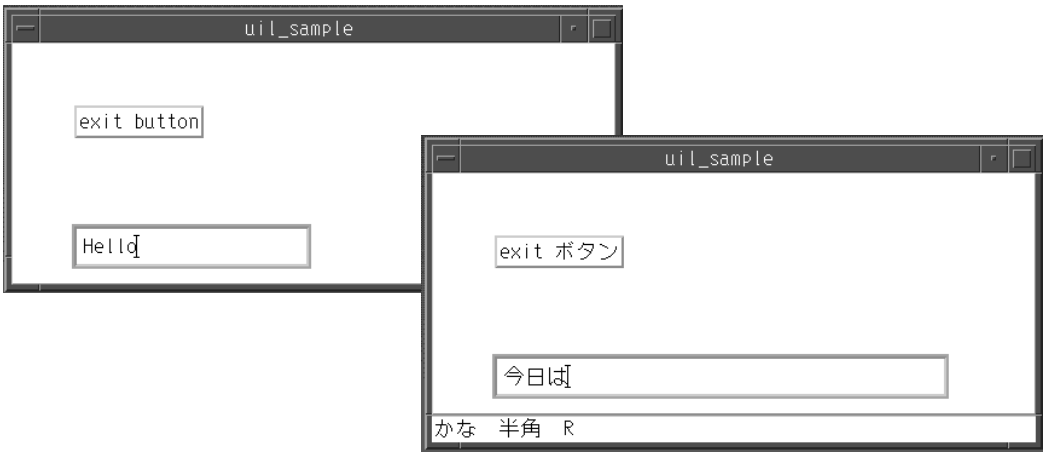

図 4–4　英語と日本語の環境での UIL のプログラム例

次のプログラム例で *LLL* はロケール・テキストを示します。*LLL* には日本語、韓国語、中国語 (繁体字)、ギリシャ語、フランス語、その他の言語が入ります。

```
uil_sample.uil
!
!        sample uil file - uil_sample.uil
!
!          C source file - uil_sample.c
!
!          Resource file - uil-sample.resource
!
module  Test
        version = 'v1.0'
```

```
        names = case_sensitive
        objects = {
            XmPushButton = gadget;
        }
!***********************************
!       declare callback procedure
!***********************************
procedure
        exit_CB ;
!*****************************************************************
!       declare BulletinBoard as parent of PushButton and Text
!*****************************************************************
object
        bb : XmBulletinBoard {
          arguments{
              XmNwidth  = 500;
              XmNheight = 200;
          };
          controls{
              XmPushButton pb1;
              XmText       text1;
          };
        };
!****************************
!       declare PushButton
!****************************
object
        pb1 : XmPushButton {
          arguments{
              XmNlabelString = #Normal ``LLLexit buttonLLL'';
              XmNx = 50;
              XmNy = 50;
          };
          callbacks{
              XmNactivateCallback = procedure exit_CB;
          };
        };
!*********************
!       declare Text
!*********************
        text1 : XmText {
          arguments{
              XmNx = 50;
              XmNy = 150;
          };
        };

end module;

/*
 *      C source file - uil_sample.c
 *
 */
#include <Mrm/MrmAppl.h>
#include <locale.h>
void exit_CB();
static     MrmHierarchy         hierarchy;
static     MrmType              *class;

/*****************************************/
```

```
/*      specify the UID hierarchy list    */
/*****************************************/
static     char                 *aray_file[] =
                                       {``uil_sample.uid''};
static     int                  num_file =
                                       (sizeof aray_file / sizeof aray_file[0]);
/*****************************************************/
/*      define the mapping between UIL procedure names*/
/*      and their addresses                          */
/*****************************************************/
static     MRMRegisterArg       reglist[]={
                  {``exit_CB'',(caddr_t) exit_CB
           }
```

UIL のコンパウンド・ストリング

UIL ファイルに文字列を指定する機構は 3 つあります。

- 文字列リテラルとして。文字列リテラルは、ヌルで終了する文字列またはコンパウンド・ストリングのいずれかとして UID ファイルに格納される可能性があります。
- コンパウンド・ストリングとして。
- ワイド文字の文字列として。

文字列リテラルとコンパウンド・ストリングは両方とも、テキスト、文字セット、描画方向から成ります。方向が明示されていない文字列リテラルとコンパウンド・ストリングについては、UIL は文字セットから描画方向を推定します。UIL 連結演算子 (&) は、文字列リテラルとコンパウンド・ストリングの両方を連結します。

UIL が文字列リテラルを UID ファイルにヌルで終了する文字列として格納するかコンパウンド・ストリングとして格納するかに関係なく、UIL は各文字列の文字セットと描画方向に関する情報をテキストと共に格納します。一般に、次のような場合に、UIL は文字列リテラルまたは文字列表現をコンパウンド・ストリングとして UID ファイルに格納します。

- 文字列表現が異なる文字セットまたは描画方向の 2 つ以上のリテラルから成る場合
- リテラルまたは文字列表現がコンパウンド・ストリングのデータ型を持つ値 (データ型がコンパウンド・ストリングであるリソースの値など) として使用される場合

UIL は、文字セットを指定する数多くのキーワードを認識します。UIL は、解析方向と文字が 8 ビットと 16 ビットのいずれかなどの解析規則を、認識する文字セットのそれぞれに関連付けます。UIL の `CHARACTER_SET` を使用して文字セットを定義することも可能です。

文字列リテラルの形式は次のいずれかです。

- *'*[*character_string*]*'*
- [*#char_set*]
- *"*[*character_string*]*"*

各形式において、文字列の文字セットは次のように決定されます。

- *'character_string'* として宣言された文字列の場合、UIL コンパイル環境に LANG 環境変数が設定されていれば、文字セットは LANG 環境変数のコード・セット・コンポーネントです。あるいは、LANG 環境変数が設定されていないかコード・セットがない場合は、文字セットは XmFALLBACK_CHARSET の値です。デフォルトでは XmFALLBACK_CHARSET の値は ISO8859-1 ですが、ベンダが別の値を供給する場合もあります。
- *#char_set "string"* として宣言された文字列の場合、文字セットは *char_set* です。
- *"character_string"* として宣言された文字列の場合、文字セットはモジュールに CHARACTER_SET 節があるかどうか、また UIL コンパイラの use_setlocale_flag が設定されているかどうかに依存します。
  - モジュールに CHARACTER_SET 節がある場合は、文字セットはその節に指定されたものになります。
  - モジュールに CHARACTER_SET 節がないが uil コマンドが -s オプションで開始された場合、または Uil() 関数が use_setlocale_flag が設定された状態で開始された場合は、UIL は setlocale() 関数を呼び出して現在のロケールの文字列を解析します。結果となる文字列の文字セットは、XmFONTLIST_DEFAULT_TAG です。
  - モジュールに CHARACTER_SET 節がなく、uil コマンドが -s オプションなしで実行された場合、または Uil() 関数が use_setlocale_flag なしで実行された場合は、UIL コンパイル環境に LANG 環境変数が設定されていれば、文字セットは LANG 環境変数のコード・セット・コンポーネントです。あるいは、LANG 環境変数が設定されていないかコード・セットがない場合は、文字セットは XmFALLBACK_CHARSET の値です。

UIL は常に、COMPOUND_STRING 関数を使用して指定された文字列をコンパウンド・ストリングとして格納します。この関数は、文字セットの文字列表現とオプション指定、方向、文字列にセパレータを追加するかどうかについてを引き数に取ります。文字セットまたは方向が指定されない場合、前の部分で説明したように UIL はその値を文字列表現から獲得します。

注 - \ (バックスラッシュ) で始まるあらかじめ定義された特定のエスケープ・シーケンスは、次の例外を除いて文字列リテラルに表示できます。

- 単一引用符に囲まれた文字列は、複数の行に渡ることが可能です。各改行文字はバックスラッシュでエスケープされます。二重引用符に囲まれた文字列は、複数の行に渡ることはできません。
- エスケープ・シーケンスは、現在のロケールで解析された文字列 (ローカライズされた文字列) の中で逐語的に処理されます。

第 5 章

Xt 依存性と Xlib 依存性

この章では、Xt および Xlib における国際化対応に関連する作業について説明します。

ロケール管理

この節では、すべてのロケール依存の Xlib 関数と共通デスクトップ環境関数をコントロールするロケール機構のサポートを定義します。

X ロケール管理

X ロケールは、ホスト環境に定義された 1 つ以上のロケールをサポートします。Xlib は ANSI (米国規格協会) C ライブラリに準拠しており、ロケール通知は `setlocale()` 関数で行います。`setlocale()` 関数は、ホスト C ライブラリと Xlib の両方のロケール・オペレーションを構成します。Xlib のオペレーションは `LC_CTYPE` カテゴリに管理されます。これを現在のロケールと言います。

XSupportsLocale() 関数は、現在のロケールが X によってサポートされているかどうかを判別するのに使用します。

クライアントは、ロケールと X モディファイアを選択する責任があります。クライアントは、ユーザがクライアントの起動時のロケール選択を無効にできる手段を提供すべきです。ほとんどのシングル・ディスプレイ X クライアントは、X とホストの処理環境の両方で、単一のロケールでオペレーションを行います。単一ディスプレイ X クライアント
は、setlocale()、XSupportsLocale()、XSetLocaleModifiers() の 3 つの関数を呼び出すことによりロケールを構成します。

X 国際化対応機能の特定のカテゴリのセマンティクスは、モディファイアを設定することで構成できます。モディファイアは実装に依存するロケール固有の文字列で命名されます。この機能の現在の唯一の標準使用方法は、キーボード入力メソッドのいくつかのスタイルの中から 1 つを選択することです。

XSetLocaleModifiers() 関数は、現在のロケールの Xlib ロケール・モディファイアを構成するために使用されます。

ロケールとモディファイアを初期化するクライアントの推薦するプロシージャが、ロケールとモディファイアを通知する方法を、以下のソースのうちの 1 つから別々に取得します (数字は優先順位を示します)。

1. コマンド行オプション
2. リソース
3. 空の文字列 (“ ”)

定義された最初のものが使用されます。

注 - ロケール・コマンド行オプション、またはロケール・リソースが定義された場合、その影響として、ローカル・ホスト環境のカテゴリ固有の設定をすべて無効にして、指定したロケールにすべてのカテゴリを設定するべきです。

ロケール依存性とモディファイア依存性

国際化対応の Xlib 関数は、ホスト環境により構成された現在のロケール
で、XSetLocaleModifiers() 関数により設定された X ロケール・モディファイアにおいて機能するか、その関数に供給される何らかのオブジェクトの作成時に構成されたロケールとモディファイアにおいて機能します。表 5–1 では、それ

ぞれのロケール依存の関数についてロケール依存性とモディファイア依存性を示しています。

表 5–1 ロケール依存性とモディファイア依存性

ロケール	影響を受ける関数	対象
	ロケール照会／構成	
setlocale	XSupportsLocale	照会されたロケール
	XSetLocaleModifiers	変更されたロケール
	リソース	
setlocale	XrmGetFileDatabase	Xrm データベースのロケール
	XrmGetStringDatabase	
XrmDatabase	XrmPutFileDatabase	Xrm データベースのロケール
	XrmLocaleOfDatabase	
	標準属性の設定	
setlocale	XmbSetWMProperties	供給されて返されたテキスト (環境ロケールの WM_property テキストのいずれか)のエンコーディング
setlocale	XmbTextPropertyToTextList	供給された／返されたテキストのエンコーディング
	XwcTextPropertyToTextList	
	XmbTextListToTextProperty	
	XwcTextListToTextProperty	
	テキスト入力	
setlocale	XOpenIM	XIM 入力メソッド.

表 5–1　ロケール依存性とモディファイア依存性　続く

ロケール	影響を受ける関数	対象
XIM	XCreateIC	XIC 入力メソッド構成
	XLocaleOfIM, など	照会されたロケール
XIC	XmbLookupText	キーボード配置
	XwcLookupText	返されたテキストのエンコーディング
	テキスト描画	
setlocale	XCreateFontSet	XFontSet のフォントの Charset
XFontSet	XmbDrawText,	供給されたテキストのロケール
	XwcDrawText, など	供給されたテキストのロケール
	XExtentsOfFontSet, など	ロケール依存メトリクス
	XmbTextExtents,	
	XwcTextExtents, など	
	Xlib エラー	
setlocale	XGetErrorDatabaseText	エラー・メッセージのロケール
	XGetErrorText	

クライアントは、X の関数と C ライブラリ関数の呼び出しのロケールが同じである場合は、X の関数に返されたロケール・エンコード・テキスト文字列が C ライブラリ関数に渡されることが可能である、または C ライブラリ関数の結果の文字列が X の関数に渡されることが可能であると想定しています。

ロケールのエンコーディングが状態に依存する場合は、国際化対応 Xlib の関数によって処理されるすべてのテキスト文字列は、ロケールのエンコーディングの初期状態で開始すると見なされます。Xlib の関数はすべて、現在のロケールや X モディファイアの設定を変更しないかのように動作します (これはつまり、Xlib またはアプリケーションのいずれかによってライブラリ内で提供され、ロケールを変更するか XSetLocaleModifiers() 関数をヌルでない引き数で呼び出す関数はすべて、入力時と終了時に現在のロケール状態を保存および復元しなければならない

という意味です)。また、ANSI C ライブラリに準拠する実装の Xlib の関数は、ANSI C の関数 mblen()、mbtowc()、wctomb()、strtok() に関連付けられたグローバルな状態を変更しません。

Xt ロケール管理

Xt ロケール管理には次の 2 つの関数が含まれます。

- XtSetLanguageProc()
- XtDisplayInitialize()

XtSetLanguageProc

Xt ツールキットの初期化前に、アプリケーションは通常 XtSetLanguageProc() 関数を次のうちどれか 1 つの形式で呼び出します。

```
XtSetLanguageProc (NULL, NULL, NULL)
```

注 - ロケールは、(たとえば XtAppInitialize() 関数を介して) ツールキットが初期化されるまでは、実際には設定されません。したがって、XtSetLanguageProc() 関数と、ツールキットの初期化の後に (たとえば catopen() 関数を呼び出す場合)、setlocale() 関数が必要になるかもしれません。

リソース・データベースは現在のプロセスのロケールに作成されます。画面ごとのリソース・データベースを作成する前のディスプレイ初期化の間に、イントリンシクス関数は、コマンド行かディスプレイごとのリソース指定にあるオプションに応じてロケールを設定するために、指定されたアプリケーション・プロシージャを呼び出します。

アプリケーションにより提供されるコールアウト・プロシージャの型は XtLanguageProc で、形式は次のとおりです。

```
typedef String(*XtLanguageProc)(displayID,languageID, clientdata);
Display *displayID;
String languageID;
XtPointer clientdata;
```

displayID ディスプレイを渡します。

languageID コマンド行またはサーバのディスプレイごとのリソース指定から取得される初期言語値を渡します。

clientdata XtSetLanguageProc() 関数への呼び出しで指定された追加のクライアント・データを渡します。

言語プロシージャにより、アプリケーションは、ロケールを XtDisplayInitialize() 関数で決定された言語リソースの値に設定できます。この関数は、以降にリソース・ファイルを読み込むために XtDisplayInitialize() 関数が使用する新しい言語文字列を返します。

最初は、イントリンシクス関数によって言語プロシージャは設定されていません。XtDisplayInitialize() 関数によって使用されるように言語プロシージャを設定するには、XtSetLanguageProc() 関数を使用してください。

```
XtLanguageProc XtSetLanguageProc(applicationcontext, procedure, clientdata)
XtAppContext applicationcontext;
XtLanguageProc procedure;
XtPointer clientdata;
```

applicationcontext 言語プロシージャが使用されるアプリケーション・コンテキストか、またはヌルの値を指定します。

procedure 言語プロシージャを指定します。

clientdata 言語プロシージャが呼び出されたときに言語プロシージャに渡される追加のクライアント・データを指定します。

XtSetLanguageProc() 関数は、XtDisplayInitialize() 関数から呼び出される言語プロシージャを、指定されたアプリケーション・コンテキストで初期化される以降のすべてのディスプレイのために設定します。*applicationcontext* パラメータがヌルの場合、指定された言語プロシージャは、呼び出しプロセスによって作成されたすべてのアプリケーション・コンテキスト (将来作成されるかもしれないアプリケーション・コンテキストも含みます) に登録されます。*procedure* パラメータがヌルの場合、デフォルトの言語プロシージャが登録されます。XtSetLanguageProc() 関数は、前に登録されていた言語プロシージャを返します。言語プロシージャが登録されていなかった場合の戻り値は不定です。しか

し、その戻り値が XtSetLanguageProc() 関数以降の呼び出しに使用される場合は、デフォルトの言語プロシージャが登録されることになります。

デフォルトの言語プロシージャは次のことを行います。

- 環境に応じてロケールを設定します。ANSI C システムでは、これは setlocale (LC_ALL, "*language*") 関数を呼び出すことで実行されます。エラーが発生した場合は、XtWarning() 関数で警告メッセージが発行されます。
- 現在のロケールがサポートされているかを確認するために、XSupportsLocale() 関数を呼び出します。サポートされていない場合、XtWarning() 関数で警告メッセージが発行され、ロケールは C に設定されます。
- 空の文字列を指定する XSetLocaleModifiers() 関数を呼び出します。
- 現在のロケールの値を返します。ANSI C システムでは、これは setlocale (LC_CTYPE, NULL) 関数への最終呼び出しからの戻り値です。

クライアントはこの機構を利用して、次の例のように XtDisplayInitialize() 関数の前に XtSetLanguageProc() 関数を呼び出すことにより、ロケールを確立できます。

```
Widget top;
XtSetLanguageProc(NULL, NULL, NULL);
top = XtAppInitialize( ... );
...
```

XtDisplayInitialize

XtDisplayInitialize() 関数は、まず指定されたディスプレイに使用される言語文字列を決定し、ディスプレイとホストとアプリケーションの組み合わせのためのアプリケーションのリソース・データベースを、次のソースから読み込みます (数字は優先順位を示します)。

1. アプリケーション・コマンド行 (argv)
2. ローカル・ホストのホストごとのユーザ環境リソース・ファイル
3. サーバのリソース属性またはローカル・ホストのユーザ選択リソース・ファイル
4. ローカル・ホストのアプリケーション固有ユーザ・リソース・ファイル
5. ローカル・ホストのアプリケーション固有クラス・リソース・ファイル

XtDisplayInitialize() 関数は、指定された各 *display* パラメータに対して固有のリソース・データベースを作成します。データベースが作成されると、*display* パラメータの言語文字列が以下のアクションのシーケンスと同じ方法で決定されます。

XtDisplayInitialize() 関数は、まず一時的なデータベースを 2 つ作成します。1 番目のデータベースはコマンド行を解析することにより構築されます。2 番目のデータベースは XResourceManagerString() 関数が返す文字列から構築されるか、あるいは XResourceManagerString() 関数がヌルの値を返す場合は、ユーザのホーム・ディレクトリのリソース・ファイルの内容から構築されます。このようなユーザ選択リソース・ファイルの名前は $HOME/.Xdefaults です。

コマンド行から構築されるデータベースが、リソース *name*.xnlLanguage とクラス *class*.XnlLanguage で照会されます。この *name* と *class* は指定されたアプリケーション名とアプリケーション・クラスです。このデータベース照会が失敗した場合、サーバのリソース・データベースが照会されます。この照会も失敗した場合は、言語は環境から決定されます。環境からの言語の決定は、LANG 環境変数の値を取り出すことで実行されます。言語文字列が見つからない場合は、空の文字列が使用されます。

アプリケーション固有のクラス・リソース・ファイル名は、アプリケーションのクラス名から構築されます。アプリケーションのクラス名は、アプリケーションのインストール時に通常サイト・マネージャによってインストールされるローカライズされたリソース・ファイルを指します。このファイルは、XtResolvePathname() 関数をパラメータ (*displayID*, *applicationdefaults*, NULL, NULL, NULL, NULL, 0, NULL) で呼び出すと見つけられます。このファイルはアプリケーションが正しく機能するのに必要な場合があるので、アプリケーション開発者が提供すべきです。クラス・リソース・ファイルがないときにリソースのセットを最小限しか必要としないシンプルなアプリケーションは、XtAppSetFallbackResources() 関数でフォールバック・リソース指定を宣言できます。

アプリケーション固有のユーザ・リソース・ファイル名はユーザ固有のリソース・ファイルを指し、アプリケーションのクラス名から構築されます。このファイルは、アプリケーションによって所有され、ふつうユーザのカスタマイズを格納しています。このファイルは、XtResolvePathname() 関数をパラメータ (*displayID*, NULL, NULL, NULL, *path*, NULL, 0, NULL) で呼び出すと見つけられます。この *path* はオペレーティング・システム固有の方法で定義されます。*path* 変数は、XUSERFILESEARCHPATH 環境変数が設定されている場合は、その値に定義されます。そうでない場合は、デフォルトはベンダの定義した値です。

結果のリソース・ファイルが存在する場合は、そのファイルはリソース・データベースにマージされます。このファイルはアプリケーションと共に提供されるか、またはユーザによって作成されます。

言語の決定の間にサーバのリソース属性かユーザ・リソース・ファイルから作成される一時的なデータベースが、リソース・データベースにマージされます。サーバ

のリソース・ファイルは完全にユーザによって作成されるので、ディスプレイに依存しないユーザ選択とディスプレイ固有のユーザ選択が含まれています。

ユーザの環境リソース・ファイルが存在する場合は、それがリソース・データベースに読み込まれ、マージされます。このファイル名はユーザおよびホスト固有です。ユーザの環境リソース・ファイル名は、絶対パス名としてユーザの XENVIRONMENT 環境変数の値から構築されます。この環境変数が存在しない場合、XtDisplayInitialize() 関数はユーザのディレクトリで .Xdefaults-*host* ファイルを検索します。この *host* はアプリケーションが実行されるマシン名です。結果のリソース・ファイルが存在する場合は、リソース・データベースにマージされます。環境リソース・ファイルには、サーバのリソース・ファイルでそれらのユーザ選択指定を補足するプロセス固有のリソース仕様が指定されることが予想されます。

フォント管理

国際化対応テキストの描画は、テキストのロケールの必要に応じて、1 つ以上のフォントのセットを使用して行われます。

システム環境内の国際化対応描画の方法は 2 つあり、それによりクライアントは静的な出力ウィジェット (たとえば XmLabel) の中から 1 つを選択したり、またはその他のプリミティブ関数で描画を行うために DrawingArea ウィジェットを選択することができます。

静的な出力ウィジェットは、テキストの XmString への変換を必要とします。

次に、Xlib ルーチンと Xlib 関数を使用してフォントを管理する機構を説明します。

フォント・セットの作成および解放

Xlib 国際化対応テキストの描画は、テキストのロケールの必要に応じて、1 つ以上のフォントのセットを使用して行われます。フォントは、クライアントとロケールに必要な charset によって供給されるベース・フォント名のリストにしたがって読み込まれます。XFontSet はオペーク型です。

- XCreateFontSet() 関数は、国際化対応テキスト描画フォント・セットを作成するために使用します。

- XFontsOfFontSet() 関数は、XFontStruct 構造体と、XFontSet に指定された全フォント名のリストを取得するために使用します。
- ベース・フォント名リストと、XFontSet に指定された選択されたフォント名リストを取得するために、XBaseFontNameListOfFontSet() 関数を使用します。
- XFontSet に指定されたロケール名を取得するために、XLocaleOfFontSet() 関数を使用します。
- XLocaleOfFontSet() 関数は、指定された XFontSet に指定されたロケールの名前を、ヌルで終了する文字列として返します。
- XFreeFontSet() 関数は、指定されたフォント・セットを解放します。関連付けられたベース・フォント名リスト、フォント名リスト、XFontStruct リスト、および XFontSetExtents が (存在する場合は) 解放されます。

フォント・セット・メトリクスの取得

国際化対応テキスト描画関数のメトリクスは、標準描画方向によって定義されます。標準描画方向とは、文字列の起点にある文字が次の文字へと進んでいくデフォルトの方向です。現在 Xlib インタフェースは、左から右の標準描画方向だけをサポートするよう定義されています。描画の起点は、テキストを描画するときに描画関数へ渡される位置です。ベースラインは、描画の起点を通って標準描画方向と並行に描画される線です。文字インクは、フォアグラウンド・カラーに塗られるピクセルであり、行間または文字間のスペースやイメージ・テキストのバックグラウンド・ピクセルは含みません。

描画関数によりテキスト方向の暗示的なコントロールが許されており、ロケール固有の文字列の字句的な分析に応じて標準描画方向に沿って文字が受け渡しされる順序を逆にすることができます。

文字の受け渡し順に関係なく、すべての文字の起点は、描画の起点の標準描画方向側にあります。特定の文字イメージの画面位置は、XmbTextPerCharExtents() 関数または XwcTextPerCharExtents() 関数で決定されます。

描画関数は、コンテキストに依存した受け渡しを実現できます。コンテキストに依存した受け渡しにおいて、文字列のために描画されるグリフは、単に個々の文字を表すグリフの組み合わせではありません。XmbDrawString() 関数で描画される 2 文字の文字列は、その 2 文字が XmbDrawString() 関数への別々の呼び出しで描画された場合とは異なる受け渡しの方法を取ります。クライアントが以前に描画され

た文字列に文字を追加または挿入する場合、正しい受け渡しを得るために、クライアントは隣接する文字をいくつか再描画する必要があるかもしれません。

描画関数は、改行文字、タブその他の制御文字を認識しません。非印刷文字 (スペースを除く) の描画時の動作は実装に依存します。テキストの流れの中で制御文字を解釈するのはクライアントの責任です。

コンテキストに依存した受け渡しを見つけ出すには、XContextDependentDrawing() 関数を使用します。XExtentsOfFontSet() 関数は、XFontSet を指定された最大エクステント構造体を取得します。XmbTextEscapement() 関数と XwcTextEscapement() 関数は、値として指定されたテキストのピクセル単位のエスケープを取得します。XmbTextExtents() 関数と XwcTextExtents() 関数は、文字列のイメージの総合割り当てボックスとロジック割り当てボックスを取得します (それぞれの引き数は *overall_ink_return* と *overall_logical_return*)。XmbTextPerCharExtents() 関数と XwcTextPerCharExtents() 関数は、指定されたフォント・セットに読み込まれたフォントを使用して、指定されたテキストの各文字の寸法を返します。

ローカライズされたテキストの描画

この節で定義される関数は、描画範囲内の指定された位置にテキストを描画します。そのような関数は、単一のフォントではなくフォント・セットと共に動作する点と、文字列のバイトを直接フォント索引として処理するのではなくフォント・セットのロケールに基づいてテキストを解釈する点を除いて、XDrawText()、XDrawString()、XDrawImageString() 関数に似ています。BadFont エラーが生成された場合、障害になっている文字の前の文字までが描画されています。

テキストは、指定されたフォント・セットにロードされたフォントを使用して描画されます。グラフィック・コンテキスト (GC) のフォントは無視され、関数によって変更されることがあります。すべてのフォントが、ある幅の規則に準拠するかどうかの妥当性検査は行われません。

指定内で描画可能な複数のフォント・セットを使ってテキストを描画するには、XmbDrawText() 関数か XwcDrawText() 関数を使用してください。指定された drawable 内で単一のフォント・セットを使ってテキストを描画するには、XmbDrawString() 関数か XwcDrawString() 関数を使用してください。指定された drawable 内で単一のフォント・セットを使ってイメージ・テキストを描

画するには、XmbDrawImageString() 関数か XwcDrawImageString() 関数を使用します。

ローカライズされたテキストの入力

次に、国際化対応テキストの入力に使用する Xlib とデスクトップの機構について説明します。Motif の Text[Field] ウィジェットを使用している場合、またはテキスト入力に XmIm API を使用している場合は、この節でバックグラウンド情報が提供されます。しかし、この節の説明はアプリケーションの設計やコードの実行には影響しません。文字入力が低レベルの Xlib 呼び出しでキーボードからどのように処理されるかに興味がない場合は、127ページの「ローカライズされたテキストのクライアント間通信規約」に進んでください。

Xlib 入力メソッドの概要

この節は、国際化対応テキスト入力に関して使用される用語と概念の定義、また Xlib が提供する機構の、想定されている使用方法の簡単な概要を提供します。

世界の多数の言語が、語を形成するために、記号 (文字) の小規模なセットから成るアルファベットを使用します。アルファベット言語でテキストをコンピュータに入力するために、ユーザは通常アルファベットに対応するキー記号の付いたキーボードを持っています。場合によっては、アルファベット言語のうち少数の文字がキーボードにないことがあります。ラテンアルファベットに基づく言語を話す多くのコンピュータ・ユーザは、英語ベースのキーボードしか持っていません。キーボード上に直接存在しない文字を入力するには、複数のキーを組み合わせて押す必要があります。そのような文字を入力するために、ヨーロッパの入力メソッド、構成入力メソッド、デッドキー入力メソッドなどで知られる多数のアルゴリズムが開発されました。

日本語は、音声記号のセットを持つ言語の一例です。各音声記号は特定の音を表します。日本語には音声記号のセットが 2 つあります (カタカナとひらがな)。一般に、カタカナは外来語の表記に使用し、ひらがなは通常の日本語の表記に使用します。この 2 つのシステムはまとめて仮名と呼ばれます。ひらがなは 83 文字、カタカナは 86 文字あります。

韓国語にも、ハングルと呼ばれる音声記号のセットがあります。基礎となる 24 の音声記号 (子音 14、母音 10) は、それぞれが特定の音を表します。1 つの音節は 2 ～ 3

の部分から成ります (最初の子音、母音、任意で最後の子音)。ハングルでは、音節をテキスト処理の基本単位として使用することが可能です。たとえば、削除操作は音声記号または音節単位に行うことができます。韓国語のコード・セットにはこのような音節が数千あります。ユーザは、入力したい語の音節を形成する音声記号を入力します。ディスプレイは各音声記号が入力されるにつれて変化するかもしれません。たとえば、ある音節の 2 番目の音声記号が入力されると、1 番目の音声記号の形とサイズが変わることがあります。同様に、3 番目の音声記号が入力されると、前の 2 つの音声記号の形とサイズが変わることがあります。

すべての言語がアルファベットか音声システムだけに頼っているわけではありません。日本語と韓国語を含む一部の言語は、表意文字による記述システムを採用しています。表意文字システムでは、記号の小規模なセットを使用してそれらの記号を組み替えて複数の語を作成するのではなく、それぞれの語が 1 つ (または複数) の一意の記号から成ります。そのような記号は非常に多く、中国語の表意文字システムである漢字では約 50,000 が識別されます。

コンピュータでの表意文字システムの使用には主に 2 つの考慮しなければならない点があります。第一に、日本、中国、韓国の標準コンピュータ文字セットには約 8,000 の文字があり、台湾の場合は 15,000 ～ 30,000 の文字があり、そのような文字は、1 文字を表現するのに 2 バイト以上が必要だということです。第 2 に、指定された言語のすべての表意文字を網羅するキーボードを用意することは明らかに不可能であり、したがって、適正な数のキーを持つキーボードを使用できるような文字入力の機構が必要になるということです。通常、そのような入力メソッドは音声体系に基づきますが、文字のグラフィカル属性に基づく方法もあります。

日本では仮名と漢字の両方を使用します。韓国ではハングルと時々ハンジャを使用します。これから、日本、韓国、中国、台湾での表意文字の入力について考察します。

日本では、仮名か英文字のいずれかを入力し、漢字に変換するための範囲を (時には自動的に) 選択します。複数の漢字が同じ音声表現を持つ場合もあります。そのような場合は、文字列を入力すると文字のメニューが表示され、ユーザは適切な候補を選択しなければなりません。選択の必要がない場合やすでに希望する表現である場合は、入力メソッドはただちに置換を実行します。ラテン文字が仮名または漢字に変換されることをローマ字変換と呼びます。

韓国では、通常は韓国語のテキストをハングル形式だけにすることが可能ですが、ハンジャ起源の語はハングルでなくハンジャで記述することを選ぶ人々もいます。ハングルからハンジャへ変換するには、変換の範囲を選択し、そのあとユーザは日本語のところで説明したのと同じ基本的な方法を取ります。

日本と韓国には広く普及した音声学上の記述システムがあるため、それらの国々で表意文字をコンピュータに入力する方法はかなり標準化されています。キーボード・キーには英文字と音声記号が書いてあり、ユーザはその 2 つのセットを切り替えることができます。

中国語の場合は状況が異なります。当局が奨励するピンインと呼ばれる音声システムがありますが、中国語のテキスト入力メソッドには統一されたものがありません。中国語の音声学的な分解 (ピンインその他) を使用するベンダもあれば表意的な分解を使用するベンダもあり、さまざまな処理系とキーボード配列が存在します。知られている手法は約 16 種類ありますが、どれも明確な標準ではありません。

また実際には、繁体字 (伝統的な中国文字) と簡体字という 2 種類の表意文字セットが使用されています。数年前、中華人民共和国は一部の表意文字を簡素化して全体的に無駄を排除するキャンペーンに着手しました。この方針の下に、文字は 5 年ごとに簡素化されます。文字はすでに何回か改訂されており、その結果として規模が縮小して単純になったセットが簡体字を形成しています。

入力メソッドのアーキテクチャ

前の節に示したとおり、数多くの異なる入力メソッドが今日使用されており、それぞれは言語、文化、歴史によって変化します。多くの入力メソッドで共通する機能は、ユーザは複数のキーストロークを入力して 1 つの文字 (または文字のセット) を構成できるということです。キーストロークから文字を構成するプロセスをプリエディットといいます。プリエディットには、複雑なアルゴリズムと、実質的なリソースを含む大規模な辞書が必要です。

入力メソッドには、ユーザに候補を示したり、辞書を表示したりするために、実際のキーストロークをフィードバックする 1 つ以上の領域が必要なことがあります。次に、該当する入力メソッド領域を示します。

ステータス領域	物理的なキーボード上にある発光ダイオード (LED) の論理的な拡張部として使用されます。ユーザにとって重要な入力メソッドの初期状態を表示するためのウィンドウです。ステータス領域は、テキスト・データおよびビットマップまたは、それらの組み合わせから成ります。
プリエディット領域	クライアントがデータを処理する前に使用されている言語のための中間テキストを表示するために使用されます。

補助領域

ポップアップ・メニューと、入力メソッドに必要なダイアログ・ボックスのカスタマイズに使用されます。どの入力メソッドにも複数の補助領域が存在する可能性があります。補助領域は入力メソッドにより、クライアントに依存しないで管理されます。複数の補助領域は、入力メソッドによって管理される別々のダイアログだと見なされます。

プリエディットには、さまざまなユーザの対話形式があります。次に、Xlib がサポートするプリエディット形式を示します。

OnTheSpot

データは直接アプリケーション・ウィンドウに表示されます。アプリケーション・データは、プリエディット・データが挿入点に表示されるように移動されます。

OverTheSpot

データは挿入点上に置かれたプリエディット・ウィンドウに表示されます。

OffTheSpot

プリエディット・ウィンドウはアプリケーション・ウィンドウの内側に表示されますが、挿入点ではありません。しばしば、この型のウィンドウはアプリケーション・ウィンドウの底部に置かれます。

Root

プリエディット・ウィンドウは RootWindow の子です。

移植性の高いアプリケーションには世界中のすべての言語の入力メソッドが必要だとすると、多くのコンピューティング・リソースが必要になります。それを避けるために、Xlib の設計目標は、アプリケーションが別々のプロセスに位置する入力メソッドと通信できるようにすることです。そのようなプロセスを入力サーバといいます。アプリケーションが接続するサーバは、アプリケーション起動時の環境、ユーザの言語、実際に使用されるエンコーディングに依存します。入力メソッドとの接続はロケール依存であると言われます。同様にユーザ依存でもあります。たとえば、ある言語が指定されると、ユーザは入力メソッドのユーザ・インタフェースの形式をある程度選択できます (選択肢がある場合)。

入力サーバの使用は通信のオーバヘッドを暗に意味しますが、アプリケーションは再リンクなしでマイグレーションできます。入力メソッドは、入力サーバへ通信するトークンとして、またはローカルなライブラリとして実現できます。

クライアントが入力メソッドと通信するために使用するアブストラクトは、XIM のデータ型で表されるオペーク・データ構造体です。このデータ構造体は、指定されたディスプレイに入力メソッドを開く `XOpenIM()` 関数によって返されます。このデータ構造体の以降のオペレーションは、クライアントと入力メソッドの間のすべての通信をカプセル化します。X クライアントが、入力メソッドを使用するためにネットワーキング・ライブラリや自然言語パッケージを使用する必要はありません。

1 つの入力サーバは、1 つ以上のエンコーディング・スキーマをサポートし、1 つ以上の言語に使用することができます。しかし、1 つの入力メソッドから返された複数の文字列は、常に XIM オブジェクトに関連付けられた (単一の) ロケールでエンコードされています。

入力コンテキスト

Xlib は、テキスト入力のマルチスレッド状態を管理する機能を提供します。クライアントが複数のウィンドウを使用していて、各ウィンドウには複数のテキスト入力領域があり、ユーザはそれらをいつでも切り替えられるという可能性もあります。特定の入力スレッドの状態を表すアブストラクトを入力コンテキストといいます。入力コンテキストは Xlib では XIC で表されます。図 5–1 を参照してください。

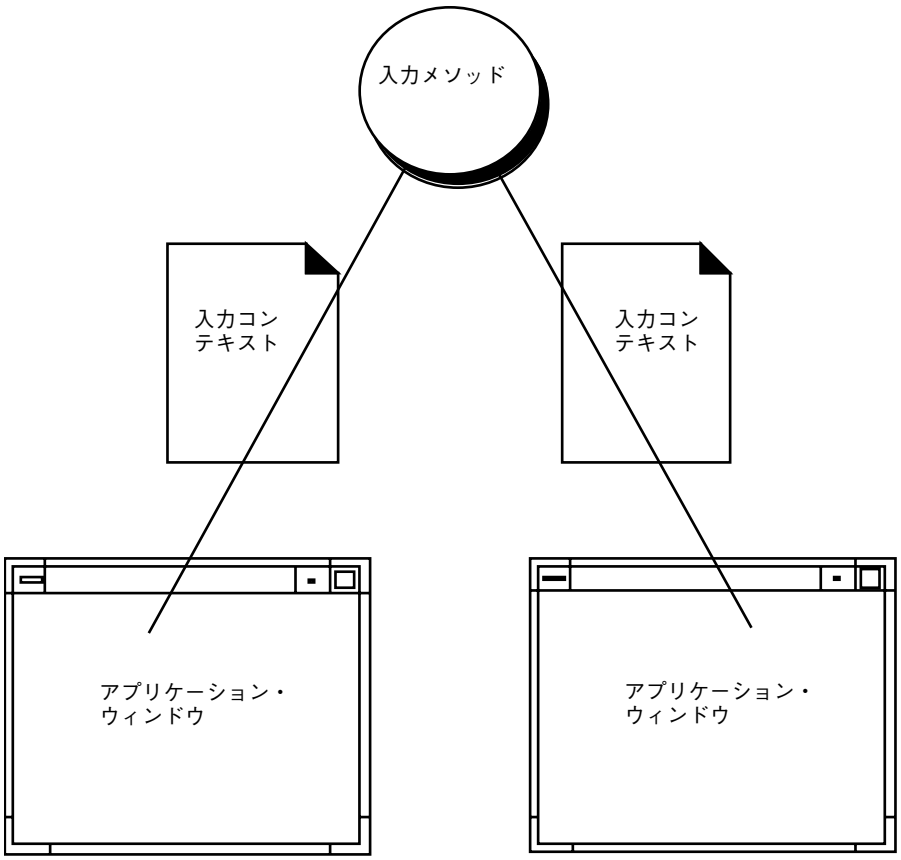

図 5–1　入力メソッドと入力コンテキスト

入力コンテキストは、クライアントと入力メソッドの間の状態、属性、通信のセマンティクスを保存するアブストラクトです。入力コンテキストは、入力メソッド、返される文字列のエンコーディングを指定するロケール、クライアント・ウィンドウ、内部状態の情報、さまざまな配置や表示の特徴の組み合わせです。入力コンテキストの概念は、グラフィック出力用にグラフィック・コンテキスト・アブストラクトが定義した入力にいくらか一致します。

1 つの入力コンテキストは、1 つの入力メソッドに属します。異なる入力コンテキストは、たぶん同じクライアント・ウィンドウで、同一の入力メソッドに関連付けることが可能です。XIC は、`XCreateIC()` 関数によって作成され、XIM 引き数を指定してそれが存在する間は入力コンテキストを入力メソッドに密接に関連付けます。入力メソッドが `XCloseIM()` 関数で閉じられる場合、密接に関連付けられてい

たどの入力コンテキストも再使用できません (入力メソッドを閉じる前に削除されることが望ましいです)。

複数のテキスト入力領域を持つクライアント・ウィンドウの例を考慮し、アプリケーションのプログラマは次の事項を選択できます。

- テキスト入力領域と同数の入力コンテキストが作成されます。クライアントはコンテキストを検索するたびに、各コンテキストに蓄積された入力を取得できます。
- アプリケーションのトップレベル・ウィンドウ用に単一のコンテキストが作成されます。そのようなウィンドウにテキスト入力領域がいくつかある場合は、ユーザが別のテキスト入力領域に移動するたびに、クライアントはコンテキストの変更を示さなければなりません。

アプリケーション設計者は、アプリケーションのニーズに応じて、入力コンテキストの範囲を単一か複数か選択できます。

キーボード入力

入力メソッドから文字を取得するには、クライアントはその入力メソッドから作成された入力コンテキストと共に `XmbLookupString()` 関数か `XwcLookupString()` 関数を呼び出さなければなりません。ロケールとディスプレイの両方とも、開かれたときに入力メソッドに割り当てられ、入力コンテキストはこのロケールとディスプレイを引き継ぎます。`XmbLookupString()` 関数または `XwcLookupString()` 関数によって返されるすべての文字列は、そのロケールでエンコードされます。

Xlib フォーカス管理

`XmbLookupString()` 関数または `XwcLookupString()` 関数が使用される各テキスト入力領域には、関連付けられた入力コンテキストがあります。

アプリケーションのフォーカスがテキスト入力領域に移動するとき、アプリケーションは、その領域に関連付けられた入力コンテキストに、入力コンテキストのフォーカスを設定しなければなりません。入力コンテキストのフォーカスは、`XSetICFocus()` 関数を適切な入力コンテキストと共に呼び出すことで設定されます。

また、アプリケーションのフォーカスがテキスト入力領域の外側へ移動するときは、アプリケーションは `XUnsetICFocus()` 関数を呼び出して、関連付けられた入力コンテキストのフォーカスを設定解除すべきです。最適化のため

に、XSetICFocus() 関数が 2 つの異なる入力コンテキストで続けて呼び出された場合、2 回目のフォーカス設定の時点で自動的に 1 回目のフォーカスが設定解除されます。

注 - 入力コンテキストの設定と設定解除を正しく行うためには、アプリケーション・レベルのフォーカスの変更を追跡する必要があります。そのようなフォーカスの変更は、必ずしも X サーバのフォーカスの変更に対応しません。

単一の入力コンテキストが複数のテキスト入力領域への入力に使用される場合は、フォーカス・ウィンドウが変更されるたびに、その入力コンテキストのフォーカス・ウィンドウを設定する必要があります。

Xlib ジオメトリ管理

ほとんどの入力メソッドのアーキテクチャにおいて (OnTheSpot は著しい例外ですが)、入力メソッドは自身のデータの表示を行います。より見やすい位置においておくために、入力メソッド領域をクライアント内に埋め込むことが望ましい場合が多くあります。このためには、クライアントが入力メソッドにスペースを割り当てる必要があるかもしれません。Xlib は、クライアントが入力メソッド領域のサイズと位置を提供できるようサポートします。ジオメトリ管理のためにサポートされている入力メソッド領域は、ステータス領域とプリエディット領域です。

入力メソッドウィンドウのジオメトリ管理の元となる基本概念は、クライアント (またはツールキット) と入力メソッドの間で責任を適切に分割することです。責任の分割は次のとおりです。

- クライアントは、入力メソッドのウィンドウのジオメトリに責任があります。
- 入力メソッドは、入力メソッドのウィンドウの内容に責任があります。また、クライアントにより指定されたジオメトリにしたがって入力メソッドのウィンドウを作成することにも責任があります。

入力メソッドはクライアントにサイズを提案することはできますが、位置を提案することはできません。入力メソッドがサイズを決定することはなく、指定されたサイズを受け入れなければなりません。

クライアントは、入力メソッドのジオメトリ管理を提供する前に、ジオメトリ管理が必要かどうかを判別しなければなりません。入力メソッドは、XGetIMValues() 関数によって返された XIMStyles 値に XIMPreeditArea() 関数か XIMStatusArea() 関数を設定することにより、ジオメトリ管理の必要性を示しま

す。クライアントは、入力メソッドへジオメトリ管理を提供することを決定するとき、XNInputStyle 値を XIC に設定することにより、その決定を示します。

クライアントが入力メソッドでジオメトリ管理を行うことを確立した後、クライアントはジオメトリを入力メソッドと交渉しなければなりません。ジオメトリは次の手順で交渉されます。

- クライアントが、領域の XNAreaNeeded 値を設定することにより、入力メソッドに領域を与えます。クライアントが入力メソッドに対して制約を持たない場合は、領域を与えないか、または幅と高さを 0 (ゼロ) に設定します。そうでない場合は、値を 1 つ設定します。
- クライアントが XIC の XNAreaNeeded 値を取得します。この値に入力メソッドが提案するサイズを返します。入力メソッドは、クライアントによって与えられるすべての制約に注意を払うべきです。
- クライアントは XIC の XNArea 値を設定して、入力メソッドのウィンドウのジオメトリを入力メソッドに示します。クライアントは、入力メソッドから要求されたジオメトリを使用しようとします。入力メソッドはこのジオメトリを受け入れなければなりません。

ジオメトリ管理を実行するクライアントは、別の IC 値を設定すると、入力メソッドが希望するジオメトリに影響が出るかもしれないことを知っておく必要があります。たとえば、XNFontSet 値と XNLineSpacing 値は、入力メソッドが希望するジオメトリを変更する可能性があります。クライアントは、必要に応じて入力メソッドのウィンドウのジオメトリを再度ネゴシエートする責任があります。

さらに、入力メソッドがジオメトリ変更を起動するための、ジオメトリ管理コールバックが提供されます。

イベント・フィルタリング

フィルタリング機構により、入力メソッドが、クライアントに透過的に X イベントを取り込めるようになっています。XmbLookupString() 関数または XwcLookupString() 関数を使用しているツールキット (またはクライアント) は、イベント処理機構のある時点で、入力メソッドに必要なイベントが確実に入力メソッドによってフィルタリングされるように、フィルタを呼び出すことになっています。フィルタがない場合、クライアントは、入力メソッドが正しく機能するために必要なイベントを受け取って破棄できます。そのようなイベントの例を次に示します。

- ローカル・モードでのプリエディット・ウィンドウ上にあるイベントをエクスポーズします。
- 入力サーバと通信するために入力メソッドはイベントを使用できます。ユーザがクライアントのコードに影響を与えたくない場合、そのような入力サーバ・プロトコル関連イベントは途中で中止しなければなりません。
- キー・イベントは、Xt が提供するような変換に割り当てられる前に、フィルタへ送信することが可能です。

クライアントは、XIC の `XNFilterEvents` 値を取得し、イベント・マスクを持つクライアント・ウィンドウのイベント・マスクに追加することを期待されます。このマスクは 0 である可能性があります。

コールバック

OnTheSpot 入力メソッドが実現される場合、クライアントだけがプリエディット・データを挿入または削除でき、場合により既存のテキストをスクロールできます。つまり、キーストロークのエコーは、入力メソッドのロジックと密接に結合され、クライアント自身によって達成されなければなりません。

キーストロークが入力されると、クライアントは `XmbLookupString()` 関数か `XwcLookupString()` 関数を呼び出します。この時点で、OnTheSpot の場合は、プリエディットでのキーストロークのエコーはまだ行われていません。入力文字を処理するクライアントのロジックに戻る前に、検索関数が、新しいキーストロークを挿入するためのエコーを行うロジックを呼び出さなければなりません。これまでに入力されたキーストロークが文字を形成する場合は、入力されたキーストロークは削除される必要があり、構成された文字が返されます。結果として、クライアント・コードによって呼び出されている間、入力メソッドのロジックは戻る前にクライアントにコールバックしなければなりません。クライアント・コード、つまりコールバック・ルーチンは、入力メソッドのロジックから呼び出されます。

入力メソッドのロジックがクライアントにコールバックしなければならない場合が数多くあります。その 1 つ 1 つの場合が、定義済みコールバック・アクションに関連付けられています。クライアントが、各入力コンテキスト別に、どのコールバックがどのアクションで呼び出されるかを指定することが可能です。

また、ステータス情報をフィードバックするためのコールバックと、入力メソッドへのジオメトリ要求を開始するコールバックも提供されます。

X サーバ・キーボード・プロトコル

この節では、サーバとキーボードのグループについて説明します。

keysym は、キーキャップ上の記号のエンコーディングです。サーバの keysym マッピングの目標は、物理的なキーボードの実際のキーキャップを反映することです。ユーザは、xmodmap コマンドを希望する新しいマッピングで実行することにより、キーボードを再定義することができます。

X Version 11 Release 4 (X11R4) では、サーバでのバイリンガル・キーボードの定義が可能です。その機能は次のとおりです。

keysym のリストは、各キー・コードに関連付けられています。対応するキーの記号のセットを説明します。

- リストが (末尾の NoSymbol エントリを除いて) 単一の keysym `K` である場合、そのリストは `K NoSymbol K NoSymbol` であるかのように処理されます。
- リストが (末尾の NoSymbol エントリを除いて) keysym `K1 K2` の対である場合、そのリストは `K1 K2 K1 K2` であるかのように処理されます。
- リストが (末尾の NoSymbol エントリを除いて) keysym `K1 K2 K3` である場合、そのリストは `K1 K2 K3 NoSymbol` であるかのように処理されます。

明示的 *void* 要素がリストで希望されているときには VoidSymbol 値が使用できます。

リストの最初の 4 つの要素は、2 つの keysym のグループに分かれます。グループ 1 には 1 番目と 2 番目の keysym が含まれ、グループ 2 には 3 番目と 4 番目の keysym が含まれます。各グループ内の 2 番目の要素が NoSymbol の場合は、そのグループは 2 番目の要素が 1 番目の要素と同じであるかのように処理されます。ただし、1 番目の要素が、小文字と大文字両方の形式が定義されたアルファベット keysym `K` である場合は例外です。そのような場合、グループは、1 番目の要素が小文字形式の `K` で 2 番目の要素が大文字形式の `K` であるかのように処理されます。

イベントから keysym を取得するための標準的な規則は、グループ 1 とグループ 2 の keysym しか使用しません。リストのその他の keysym の解釈はここでは指定されません。どのグループを使用するかはモディファイアの状態が決定します。グループ間の切り替えは、MODE SWITCH という名前の keysym でコントロールされ、その keysym をあるキー・コードに接続し、そのキー・コードを Mod1 ～ Mod5 のモディファイアのうちのいずれかに接続することで実行されます。このようなモディファイアを「グループモディファイア」と呼びます。どのキー・コードに対しても、グループ 1 はグループ・モディファイアがオフの時に使用され、グループ 2 はグループモディファイアがオンの時に使用されます。

グループ内で、使用する keysym もやはりモディファイアの状態により決定されます。1 番目の keysym は、Shift モディファイアと Lock モディファイアがオフの時に使用されます。2 番目の keysym は、Shift モディファイアがオンの時、Lock モディファイアがオンの時、2 番目の keysym が大文字のアルファベットの時、または Lock モディファイアがオンで ShiftLock と解釈される時に使用されます。そうでない場合は、Lock モディファイアがオンで CapsLock と解釈される時に、Shift モディファイアの状態は初めて keysym 選択のために適用されます。keysym が小文字のアルファベットの場合は、対応する大文字の keysym が代わりに使用されます。

ジオメトリはベンダ固有の方法で定義されるかもしれませんが、キー上の記号の空間的なジオメトリは、keysym リスト順には定義されません。サーバはキー・コードと keysym 間のマッピングを使用しません。むしろ、サーバはクライアントによる読み書きのためにだけ、マッピングを格納します。

Lock という名前の KeyMask モディファイアは、CapsLock キーか ShiftLock キーのいずれかにマップされますが、どちらにマップされるかは、アプリケーション固有の決定か、ユーザ固有の決定か、あるいはその両方に委ねられます。しかし、対応するキー・コードに関連付けられた keysym に応じて、ユーザがマッピングを決定することをお勧めします。

ローカライズされたテキストのクライアント間通信規約

次の情報は、コンポーネントがテキスト・データと通信するために ICCC (クライアント間通信規約) を使用する方法を説明し、ICCC 選択が実行される方法を理解するためのガイドラインを示します。`XmText` ウィジェット、`XmTextField` ウィジェット、`dtterm` コマンドはこのガイドラインを厳守しています。

ツールキットは国際化対応 ICCC 準拠のために拡張されています。`XmText`、`XmTextField`、および`dtterm` の選択機構は、どの選択トランザクションでもデータとデータ・エンコーディングが確実に適切に一致するように拡張されています。これには標準的なカット・アンド・ペースト操作が含まれます。

アプリケーションの記述にツールキットを使用する開発者に対して、ツールキットはアプリケーションを ICCC 準拠にすることができます。しかし、ツールキット・ベースのアプリケーションと通信するアプリケーションを開発するためにその他の ICCC に準拠していないツールキットを使用する可能性のある開発者に対しては、以下の説明が役に立ちます。

所有側の選択

どんな所有側でも、XA_TARGETS がローカライズされたテキストで要求された場合、少なくとも次のアトム・リストを返します。

- 現在のロケールのアトム・コード・セット
- COMPOUND_TEXT
- XA_STRING

XA_TEXT が要求されると、所有側はテキストを属性セットのエンコーディング型と共に現在のロケールのコード・セットへそのまま返します (データ変換は行われません)。ロケールのコード・セットの名前を表すアトムが作成されます。

COMPOUND_TEXT が要求されると、所有側はローカライズされたテキストをコンパウンド・テキストに変換し、それを属性型 COMPOUND_TEXT で渡します。

XA_STRING が要求されると、所有側はローカライズされたテキストを XA_STRING に変換しようとします。テキスト文字列の中に XA_STRING に変換できない文字がある場合は、このオペレーションは失敗します。

注 - XA_STRING は ISO8859-1 であるように定義されています。

選択の要求側

要求側は、テキスト・データが選択の所有側と通信されるとき、まず XA_TARGET を要求します。

次に、要求側は次のうち 1 つのアトムを次の優先順位で検索します。

- 要求側のロケールのコード・セットのアトム
- COMPOUND_TEXT
- XA_STRING
- XA_TEXT

要求側のロケールのコード・セットがターゲットの 1 つに一致する場合は、要求側はそのコード・セットを表すアトムを使用して要求を実行します。XA_TEXT アトムは、その他のアトムが見つからない場合にだけ使用されます。所有側が自分のエンコーディングを表す型と共に属性を返すので、要求側は自分のロケールのコード・セットに変換しようとします。

COMPOUND_TEXT または XA_STRING の型が要求された場合、要求側は XmbTextPropertyToTextList() 関数か XwcTextPropertyToTextList() 関数を使用して、テキストを要求側の現在のロケールのコード・セットに正しく変換しようとします。このような関数は、所有側クライアントと要求側クライアントが異なるコード・セットの下で実行しているときに使用されます。

COMPOUND_TEXT または XA_STRING から変換する場合、すべてのテキスト・データが変換を保証されるわけではありません。所有側と要求側の間で共通の文字だけが変換されます。

XmClipboard

XmClipboard も、XmText ウィジェットと XmTextField ウィジェットと共に、ICCC 準拠になるよう拡張されています。XmText ウィジェットと XmTextField ウィジェットを介してテキストがクリップボード上に置かれるとき、次の ICCC プロトコルが実行されます。

XmText ウィジェットと XmTextField ウィジェットを介してテキストがクリップボードから取り出されるとき、そのテキストは COMPOUND_TEXT または XA_STRING から現在のロケールのエンコーディングに変換されます。クリップボードのすべてのテキストは、コンパウンド・テキスト形式か文字列形式のいずれかであると想定されます。

注 - テキストがクリップボード上に直接置かれるとき、アプリケーションは、形式またはアトム形式のエンコーディング型を、クリップボードに置くテキストと共に指定する必要があります。同様に、テキストがクリップボードから直接取り出される場合も、取り出す側のアプリケーションは、クリップボードのデータがどのようにエンコードされているかを見るために形式を確認して、適切なアクションを取る必要があります。

ウィンドウ・マネージャへウィンドウのタイトルとアイコン名を渡す

VendorShell クラスの XtNtitleEncoding リソースと XtNiconNameEncoding リソースのデフォルトは、None に設定されます。これは libXm.a ライブラリ使用時にのみ実行されます。libXt.a ライブラリはリソースのデフォルトとして XA_STRING を維持します。

これは、テキスト (タイトルとアイコン名) がローカライズされているという前提のもと、デフォルトで XmNtitle リソースと XmNiconName リソースがコンパウンド・テキストのような標準 ICCC 形式に変換されるように実行されます。

ユーザは XtNtitleEncoding リソースと XtNiconNameEncoding リソースを設定しない方が良いと思われます。その代わりに、XtNtitle リソースと XtNiconName リソースが、必ず実行中のクライアントの現在アクティブなロケールのエンコーディングでエンコードされた文字列であるようにしてください。None 値が使用される場合は、ツールキットはローカライズされたテキストを標準の ICCC 形式に変換します (通信されるエンコーディングは COMPOUND_TEXT または XA_STRING です)。XtNtitleEncoding リソースと XtNiconNameEncoding リソースが設定されている場合は、 XtNtitle リソースと XtNiconName リソースはどんな方法でも変換されず、指定されたエンコーディングでウィンドウ・マネージャに通信されます。

通信されているウィンドウ・マネージャが ICCC 準拠だと想定すると、そのウィンドウ・マネージャはエンコーディング型 COMPOUND_TEXT か、XA_STRING か、あるいはその両方を使用できます。

XmBulletinBoard ウィジェット・クラスの XmNdialogTitle リソースを設定するとき、charset セグメントに制限があることに注意してください。X コンソーシアムの標準コンパウンド・テキスト・エンコーディングではない charset か、XmFONTLIST_DEFAULT_TAG に関連付けられていない charset の場合、テキスト・セグメントはローカライズされたテキストとして処理されます。ローカライズされたテキストは、ウィンドウ・マネージャに通信される前に、コンパウンド・テキストか、ISO8859-1 に変換されます。

ウィンドウ・マネージャは、クライアントから渡されたクライアントのタイトルとアイコン名を、現在のロケールのエンコーディングに常に変換できるように拡張されています。XmString が XmFONTLIST_DEFAULT_TAG 識別子を使用して作成されます。このように、クライアントのタイトルとアイコン名は、常にウィンドウ・マネージャのフォント・リストのデフォルトのフォント・リスト・エントリで描画されます。

注 - これにより、コード・セットは異なるが文字セットが似ているクライアントが、タイトルをウィンドウ・マネージャに通信することが可能になります。たとえば、PC コード・クライアントと ISO8859-1 クライアントの両方が、ウィンドウ・マネージャのコード・セットに関係なくタイトルを表示できます。

メッセージ

システム環境ツールキットに基づくアプリケーションを国際化対応にする作業で、アプリケーションのソース内にハードコードされたロケール固有のデータがないことが重要な場合があります。一般的なロケール固有の項目の 1 つに、標準 I/O (入出力) のアプリケーションによって返される (エラーおよび警告) メッセージがあります。

一般的に、システム環境ツールキット・ウィジェットまたはガジェットを介してユーザに表示されるエラーおよび警告メッセージの場合、メッセージはすべてメッセージ・カタログを介して外側に置く必要があります。

ツールキットのコンポーネントを介して表示されるダイアログ・メッセージの場合、メッセージはすべてローカライズされたリソース・ファイルを介して外側に置く必要があります。これは、`XmLabel` クラスと `XmPushbutton` クラスの `XmNlabelString` リソースまたはウィンドウのタイトルのようなリソースをローカライズするのと同じ方法で実行されます。

たとえば、警告メッセージが `XmMessageBox` ウィジェット・クラスを介して表示される場合、`XmNmessageString` リソースをアプリケーションのソースコード内でハードコードすることはできません。代わりに、このリソースの値はメッセージ・カタログから取り出す必要があります。異なる複数のロケールで実行すると想定された国際化対応アプリケーションの場合、サポートされる各ロケールに対して、ローカライズされた個別のカタログが存在しなければなりません。このように、アプリケーションは再構築の必要はありません。

ローカライズされたリソース・ファイルは、`/etc/dt/app-defaults/%L` サブディレクトリに入れるか、または `XENVIRONMENT` 環境変数が指すようにすることができます。*%L* 変数は、実行時に使用されるロケールを示します。

前述の 2 点の選択は、設計時に決定するものとしてアプリケーション開発者に任されています。

付録 **A**

メッセージ・ガイドライン

簡単に国際化対応にできるメッセージを作成するために、この付録にある情報を参照してください。

ファイル名の規約

この節では、ファイルにユーザ・メッセージの名前を付ける際の規約を説明します。通常、メッセージ・ソース・ファイルには拡張子 .msg が付き、生成されるメッセージ・カタログには拡張子 .cat が付きます。その他にもメッセージ関連のファイルがある場合もあります。ファイルが.msg や .cat などの拡張子を持つためには、次の基準を満たしていなければなりません。

- X/Open 準拠である

- gencat コマンドを使用することにより *.cat ファイルになる

原因および回復情報

可能な限り、何が起こったのか、またその状況を修復するために何ができるかをユーザに対して正確に説明します。

Bad arg というメッセージはあまり役に立ちません。しかし、次のメッセージなら、コマンドを機能させるためにどうすべきかを正確にユーザに伝えています。

```
Do not specify more than 2 files on the command line
```

同様に、Line too long というメッセージもユーザに回復情報を与えていません。しかし、次のメッセージは、ユーザに対して具体的な回復情報を提供します。

```
Line cannot exceed 20 characters
```

与えられたエラー・メッセージに詳しい回復情報が必要な場合は、オンライン情報やヘルプの適切な場所に追加してください。

142ページの「メッセージの例」で、元のメッセージと書き直したメッセージの例を参照してください。

翻訳者のためのコメント行

メッセージのソース・ファイルには、翻訳のプロセスで翻訳者に役立つコメントを入れることができます。そのようなコメントは、生成されるメッセージ・カタログの一部ではありません。このコメントは、C 言語でのプログラムのドキュメント化を助けるコメントに似ています。ドル記号 ($) のあとにスペース 1 つが続くと、翻訳ツールと gencat コマンドによって、コメントと認識されます。メッセージ・ソース・ファイルのコメント行の例を次に示します。

```
$ This is a comment
```

翻訳者や著者に *%s*、*%c*、*%d* などの変数が何を表すかを伝えるために、コメント行を使用します。たとえば、その変数がユーザ、ファイル、ディレクトリ、フラグなどを指すかどうかを注意書きします。

コメント行は、メッセージ・カタログの最後ではなく、コメントが参照するメッセージの下に置いてください。セット全体に関する包括的なコメントは、ソース・ファイル内で $set 指示子の下に置けます。

メッセージ・カタログ内の使用しなくなったメッセージは、コメント行に指定してください。

プログラム形式

メッセージのプログラミング形式については、次の事項を参照してください。

- メッセージを文節単位で構築しないでください。適切なときに完全なメッセージが発行できるように情報を渡すには、フラグその他の手段をプログラム内で使用してください。
- 既存のメッセージの *%s* 文字列の変数として、ハードコードされた英語のテキストを使用しないでください。これはメッセージの構文でもあり、翻訳できません。
- 文の最初の語は大文字にして、文または句の最後にはピリオドを使用します。
- メッセージの最終行は \n (バックスラッシュと小文字の n。改行を示します) で終わらせます。1 行のメッセージの場合も同様です。
- メッセージの 2 行目以降の行は \t (バックスラッシュと小文字の t。タブを示します) で始めます。
- その他のすべての行は \n\ (バックスラッシュ、小文字の n、バックスラッシュ。改行を示します) で終わらせます。
- 何らかの理由でメッセージが改行で終わらない場合は、開発者にその旨をコメントで通知します。
- 各メッセージの前に、そのメッセージを呼び出したコマンドの名前をコロンと共に付けます。エラー・メッセージではコマンド名の前にコンポーネント番号が付きます。コマンド名がメッセージ内にある場合は次のようになります。

```
OPIE ''foo: Opening the file.''
```

記述形式

メッセージの記述形式に関する次のガイドラインには、用語、句読法、叙法、態、時制、キャピタリゼーション、その他の使用法に関する項目が含まれます。

- 文の形式を使用します。1 行で 1 文のメッセージが望ましいです。
- 冠詞 (*a*、*an*、*the*) はあいまいさをなくす必要がある場合に付けます。
- 文の最初の語は大文字にして、最後にはピリオドを付けます。
- 現在形を使用します。メッセージは未来形にしないでください。たとえば次のような文を使用します。

  ```
  The foo command displays a calendar.
  ```

  次のようにはしないでください。

  ```
  The foo command will display a calendar.
  ```

- メッセージでは一人称 (*I* または *we*) を使わないでください。
- 二人称の使用も避けてください。

  ヘルプおよび対話型テキスト以外では、*you*という語は使わないでください。

- 能動態を使用します。次の例の 1 行目は元のメッセージで、2 行目は望ましい書き方です。

  ```
  MYNUM ''Month and year must be entered as numbers.''
   MYNUM ''foo: 7777-222 Enter month and year as numbers.\n''
  ```

  7777-222 はメッセージ ID です。

- 命令法 (コマンド句) と、能動態を示す動詞を使用します。たとえば、*specify*、*use*、*check*、*choose*、*wait* などです。
- メッセージは肯定的な表現にします。次の例の 1 行目は元のメッセージで、2 行目は望ましい書き方です。

  ```
  BADL ''Don't use the f option more than once.''
   BADL ''foo: 7777-009 Use the -f flag only once.\n''
  ```

- 名詞を動詞として使用しないでください。辞書にある文法の範囲内で語を使用します。ある語が辞書に名詞としてしか載っていない場合、その語は動詞として使用しないでください。たとえば、*solution a problem* (また、*architect a system*) のようには使わないでください。

- 接頭辞や接尾辞を使用しないでください。翻訳者が *re-*、*un-*、*in-*、*non-* で始まる語を解釈できないかもしれないため、このような接頭辞や接尾辞を使用したメッセージは、翻訳されると意図した意味とは異なってしまう可能性があります。しかし、接頭辞が一般的に使用される語の一部に完全になっている場合は例外です。*previous* や *premature* は使用できます。*nonexistent* は使用できません。
- 複数形は使用しないでください。*error(s)* のように括弧を使って単数と複数を示すことはしないでください。この書き方だと翻訳できません。単数と複数を示さなければならない場合は、*error or errors* としてください。よりよい方法は、状況に合わせて単数形と複数形の 2 つの異なるメッセージのいずれかが発行されるようにコードを作成することです。
- 短縮形は使用しないでください。システムが処理できないことを表すには *cannnot* だけを使用してください。
- 引用符を使用しないでください。単一引用符も二重引用符も同じです。たとえば、*%s*、*%c*、*%d* などの変数やコマンドに引用符を付けないでください。ユーザは引用符を文字どおり入力するものだと思うかもしれません。
- 行の終わりで語をハイフンで結ばないでください。
- メッセージでは一般的な強調表示の方法を取らないでください。また、単語の先頭あるいはすべての単語を大文字表記することによる強調表示の方法は用いないでください。
- *and/or* は使用しないでください。この構文は他の言語には存在しません。通常、必ずしも両方する必要がない場合は、*or* とする方が適します。
- 24 時間制を使用します。*a.m.* や *p.m.* を使って時刻を指定しないでください。たとえば、*1:00 p.m.* は *13:00* と記述します。
- 頭文字による略記は避けてください。略さずに書いたものより頭文字による略記の方がよく知られている場合にだけ略記を使用します。略記を複数形にするには、小文字の *s* をアポストロフィなしで付けます。記述する前に、それが商標でないことを確認してください。
- 「禁句」は避けてください。たとえば、*abort*、*argument*、*execute* などです。プロジェクト用語集を参照してください。
- 意味のある用語を使うよう心がけてください。メッセージが確実に意味をなし、翻訳可能であるようにする一方、元のメッセージ・テキストの意味をできるだけ維持してください。

使用方法の説明文

使用方法の説明文は、少なくとも 1 つの無効なフラグがコマンド行で指定された場合に、コマンドによって生成されます。フラグに関連付けられたデータがない場合または不正な場合は、使用方法の説明文は使用してはなりません。そのような場合は、その問題に固有のエラー・メッセージが使用されます。

- 使用方法の説明文にはコマンド形式を示します。たとえば、del コマンドの使用方法の説明文の例は次のとおりです。

```
Usage: del {File ...|-}
```

- コマンドの目的を定義する節は削除してください。
- *File*、*Directory*、*String*、*Number* などの語 (パラメータ) の最初の文字は、使用方法の説明文で使用するときだけ大文字にします。
- コマンド行でパラメータを省略形にしないでください。経験のあるユーザにとっては *Num* が *Number* であることは明らかですが、それでも確実に正しく翻訳されるように略さず書いてください。
- 使用方法の説明文では、区切り文字は次のものだけを使用します。

区切り	文字説明
[]	パラメータはオプションです。
{ }	パラメータの選択肢が 2 つ以上ありますが、そのうちの 1 つが必要です (下のテキストを参照)。
|	パラメータを 1 つだけ選択してください。[a|b] は、*a* か *b* のいずれかを選択するか、a も b もどちらも選択しなくて良いという意味です。{a|b} は、*a* と *b* のいずれかを選択しなければならないと言う意味です。
..	そのコマンド行でパラメータを繰り返せます (この記号の前にスペースが 1 つ入ることに注意)。
-	標準入力

- 使用方法の説明文パラメータは、角括弧や中括弧を必要としません。もし必要で、それが唯一の選択肢の場合は、下記のようになります。

```
banner String
```

- 使用方法の説明文では、コマンド行中で分けられなければならないフラグの間にスペースを 1 つ入れます。次に例を示します。

```
unget [-n] [-rSID] [-s] {File|-}
```

- 区切りスペースなしで複数のフラグを一緒に使用できる場合は、フラグをコマンド行でスペースによって分けないでください。次に例を示します。

```
wc [-cwl] {File ...|-}
```

- コマンド行でのフラグの順序に特に意味がない場合は、フラグをアルファベット順に並べます。大文字と小文字が混在する場合は、小文字を先にします。

```
get -aAijlmM
```

- 使用方法の説明文が長くて改行する場合があります。使用方法の説明文をどこで終わらせるかはよく考えて決めてください。次の例は、get コマンドの古い形式の使用方法の説明文です。

```
Usage: get [-e|-k] [-cCutoff] [-iList] [-rSID] [-wString] [xList] \
           [-b] [-gmnpst] [-l[p]] File ...
Retrieves a specified version of a Source Code Control System (SCCS) file.
```

標準メッセージ

POSIX.2 ドキュメンテーションに定義された標準エラーを持つコマンドがあります。当てはまる場合は、POSIX.2 でセットアップされたガイドラインに従ってください。

- キーボードのキーを選択するために、ユーザに `Press the ------ key` と告げます。`------` には押すキーが入ります (`Press Ctrl-D` 等)。
- システム負荷がかかりすぎない限り、ユーザに `Try again later` と告げる必要はありません。この意味はメッセージからも明らかです。
- メッセージ・テキストを記述する場合、コマンド行にあるテキストを表すには *parameter* (パラメータ) という語を使用し、数字データを示すには *value* (値) という語を使用します。
- *command option* (コマンド・オプション) ではなく *flag* (フラグ) という語を使用します。
- 1000 分の 1 の位で値を区切るのにカンマを使用しないでください。
- 1,000 と記述しないで、1000 としてください。

- メッセージをアスタリスクで強調しなければならない場合、メッセージの前に 2 つ、後に 2 つのアスタリスクを使用してください。

```
** Total **
```

- *log in* と *log off* は動詞として使用してください。

```
Log in to the system; enter the data; then log off.
```

- *user name*、*group name*、*login* は名詞として使用してください。

```
The user name is sam.
 The group name is staff.
 The login directory is /u/sam.
```

- ユーザ番号とグループ番号は、ユーザの名前とグループに関連付けられた番号を指します。
- *super user* (スーパーユーザ) という用語は使用しないでください。*root user* (ルート・ユーザ) はすべての特権を持っているとは限りません。
- パラメータを持つコマンドを *command string* (コマンド文字列) と呼んでください。
- 多くの同じメッセージが何度も現われます。表 A-1 に、古いメッセージを置き換える新しい標準メッセージをリストします。

表 A–1 新しい標準メッセージ

使用すべき標準メッセージ	使用すべきでないメッセージ
`Cannot find or open the file.`	`Can't open filename.`
`Cannot find or access the file.`	`Can't access`
`The syntax of a parameter is not valid.`	`syntax error`

正規表現の標準メッセージ

表 A-2 は、正規表現の標準エラーメッセージを、各正規表現エラーに関連付けられたメッセージ番号と共にリストします。

表 **A–2** 正規表現の標準メッセージ

番号	使用すべき標準メッセージ	使用すべきでないメッセージ
11	`Specify a range end point that is less than 256.`	`Range end point too large.`
16	`The character or characters between \{ and \} must be numeric.`	`Bad number.`
25	`Specify a \digit between 1 and 9 that is not greater than the number of subpatterns.`	`\digit out of range.`
36	`A delimiter is not correct or is missing.`	`Illegal or missing delimiter.`
41	`There is no remembered search string.`	`No remembered search string.`
42	`There is a missing \( or \).`	`\(\) imbalance.`
43	`Do not use \( more than 9 times.`	`Too many \(.`
44	`Do not specify more than 2 numbers between \{ and \}.`	`More than two numbers given in \{ and \}.`
45	`An opening \{ must have a closing \}.`	`} expected after \.`
46	`The first number cannot exceed the second number between \{ and \}.`	`First number exceeds second in \{ and \}.`
48	`Specify a valid end point to the range.`	`Invalid end point in range expression.`
49	`For each [ there must be a ].`	`[ ]imbalance.`
50	`The regular expression is too large for internal memory storage. Simplify the regular expression.`	`Regular expression overflow.`

メッセージの例

次に、元のメッセージと書き直したメッセージの例を示します。元のメッセージの下に書き直したメッセージがあります。

```
AFLGKEYLTRS ''Too Many -a Keyletters (Ad9)''
AFLGKEYLTRS ''foo: 7777-007 Use the -a flag less than 11 times.\n''

FLGTWICE ''Flag %c Twice  (Ad4)''
FLGTWICE ''foo: 7777-004 Use the %c header flag once.\n''

ESTAT ''can't access %s.\n''
ESTAT ''foo: 7777-031 Cannot find or access %s.\n''

EMODE ''foo: invalid mode\n''
EMODE ''foo: 7777-033 A mode flag or value is not correct.\n''

DNORG ''-d has no argument  (ad1)''
DNORG ''foo: 7777-001 Specify a parameter after the -d flag.\n''

FLOORRNG ''floor out of range  (ad23)''
FLOORRNG ''foo: 7777-021 Specify a floor value greater than 0\n\
\tand less than 10000.\n''

AFLGARG ''bad -a argument  (ad8)''
AFLGARG ''foo: 7777-006 Specify a user name, group name, or\n\
\tgroup number after the -a flag.\n''

BADLISTFMT ''bad list format  (ad27)''
BADLISTFMT ''foo: 7777-025 Use numeric version and release\
\tnumbers.\n''
```

索引

あ

い

う

え

お

か

き

し

す

せ

そ

つ

て

に