



JFP 開発ガイド

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

Part No: 805-5819-10
1998 年 11 月

本製品およびそれに関連する文書は著作権法により保護されており、その使用、複製、頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。日本サン・マイクロシステムズ株式会社による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company, Ltd. が独占的にライセンスしている米国ならびに他の国における登録商標です。フォント技術を含む第三者のソフトウェアは、著作権により保護されており、提供者からライセンスを受けているものです。

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

本製品に含まれる HG 明朝 L と HG ゴシック B は、株式会社リコーがリョーベイマジクス株式会社からライセンス供与されたタイプフェイスマスタをもとに作成されたものです。平成明朝体 W3 は、株式会社リコーが財団法人 日本規格協会 文字フォント開発・普及センターからライセンス供与されたタイプフェイスマスタをもとに作成されたものです。また、HG 明朝 L と HG ゴシック B の補助漢字部分は、平成明朝体 W3 の補助漢字を使用しています。なお、フォントとして無断複製することは禁止されています。

Sun、Sun Microsystems、SunSoft、SunDocs、SunExpress、SunOS は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

サンロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャに基づくものです。

OPENLOOK、OpenBoot、JLE は、日本サン・マイクロシステムズ株式会社の登録商標です。

Wnn は、京都大学、株式会社アステック、オムロン株式会社で共同開発されたソフトウェアです。

Wnn6 は、オムロン株式会社で開発されたソフトウェアです。(Copyright OMRON Co., Ltd. 1998 All Rights Reserved.)

ATOK は、株式会社ジャストシステムの登録商標です。

ATOK7 は株式会社ジャストシステムの著作物であり、ATOK7 にかかる著作権その他の権利は、すべて株式会社ジャストシステムに帰属します。

ATOK8 は株式会社ジャストシステムの著作物であり、ATOK8 にかかる著作権その他の権利は、すべて株式会社ジャストシステムに帰属します。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問い合わせください。

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカル・ユーザインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、このライセンスは米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

DtComboBox ウィジェットと DtSpinBox ウィジェットのプログラムおよびドキュメントは、Interleaf, Inc. から提供されたものです。(Copyright (c) 1993 Interleaf, Inc.)

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含みそれに限定されない、明示的であるか黙示的であるかを問わない、なんらの保証も行われぬものとします。

本製品が、外国為替および外国貿易管理法(外為法)に定められる戦略物資等(貨物または役務)に該当する場合、本製品を輸出または日本国外へ持ち出す際には、日本サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

© 1998 by Sun Microsystems, Inc.



目次

- はじめに v
- 1. 概要 1
 - 国際化プログラミング 1
 - 言語対応化と環境変数 1
 - 標準への準拠 2
 - 文字集合とコードセット 2
 - CSI 2
 - 日本語化プログラミングの移植性と利便性 3
- 2. 国際化 API での日本語処理 5
 - 概説 5
 - 複数バイト表現での文字列処理 6
 - プログラム例 7
 - ワイド文字表現での文字列処理 9
 - プログラム例 10
- 3. 日本語ロケールと文字分類 17
 - 概説 17
 - 日本語文字対応 18
 - プログラム例 19
 - 日本語文字分類 22

	プログラム例	23
4.	日本語文字コード変換	27
	概説	27
	複数バイト・ワイド文字の相互変換	28
	プログラム例	29
	汎用コード変換	30
	プログラム例	31
5.	メッセージ処理	35
	概説	35
	アプリケーションのメッセージ処理手順	36
	X/Open 方式	36
6.	シェルスクリプトの国際化と日本語化	39
	概説	39
	国際化に影響するロケール環境変数	39
	スクリプトのメッセージ処理	40
A.	日本語専用ライブラリ (libjapanese.a)	43
	概説	43
	ワイド文字列処理関数	44
	ワイド文字変換用関数 (jconv)	47
	ワイド文字分類関数 (jctype)	48
	x86: コード変換用関数 (jisconv 、 ibmjcode)	51
B.	Solaris の日本語 TrueType フォント	55
	TrueType フォントのサポート	55
	Solaris で提供する日本語 TrueType フォント	56
	Soralis 2.5.1 以前のシステムとのフォントの互換性	57
	Soralis 2.5.1 からの変更点	58
	TrueType フォントのインストール方法	60
	索引	63

はじめに

このマニュアルでは、Solaris™ 上で日本語の処理を行うアプリケーションを開発するため、日本語処理で使用できる国際化・日本語化プログラミングインタフェースを扱う方法について説明します。プログラミングの国際化に関する一般的な知識については、関連する Solaris のマニュアルをお読みください。なお、国際化された GUI アプリケーションの作成方法については『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)』をお読みください。

対象読者

このマニュアルでは、読者が次の項目に関する一般的な知識を持っていることを前提に、日本語処理プログラミングで有効な項目に絞って解説します。

- SunOS™ 上でのプログラミング環境 (コンパイラ、デバッガ、各種ユーティリティ)
- プログラムの国際化と言語対応化
- プログラミング言語 (本書では C 言語およびシェルスクリプトを用いて説明します)
- 文字集合とエンコーディング
- 文字コードの複数バイト表現とワイド文字表現

このマニュアルの構成

このマニュアルは次の章から構成されています。

第 1 章

本マニュアルを読み進むに当たって、必要、または知っている便利な概念について説明し、関連マニュアルを紹介します。

第 2 章

X/Open™ Portability Guide, Issue4 Release2 (以下「XPG」と記述) で仕様が決定された国際化文字列処理プログラミングインタフェースを紹介し、日本語文字列を処理する方法を説明します。

第 3 章

XPG において仕様が決定された国際化文字対応・分類プログラミングインタフェース、および Solaris で仕様が追加された日本語ロケール用文字対応・分類インタフェースを紹介し、日本語文字の分類について説明します。

第 4 章

文字列の複数バイト表現とワイド文字表現の相互変換のためのインタフェース、および `iconv()` 汎用コード変換インタフェースを紹介し、その利用方法を説明します。

第 5 章

メッセージ処理のためのインタフェースを紹介し、国際化・日本語化されたアプリケーションでの日本語のメッセージを取り扱い、表示させる方法を説明します。

第 6 章

シェルスクリプトに対するロケール環境の設定とメッセージ処理コマンドを紹介し、その利用方法を説明します。

付録 A

日本語専用ライブラリの使い方を説明します。なお、この章で説明するインタフェースは `ja` (または `japanese`) ロケール上でのみ動作が保証されます。

付録 B

Solaris で提供する日本語 TrueType フォントと、そのインストール方法について説明します。

前もって読むべきマニュアル

- 『Solaris 7 ご使用にあたって』
- 『Solaris 7 オンラインリリース情報』
- 『JFP ユーザーズガイド』

関連マニュアル・書籍

- 『*Internationalization Programmer's AnswerBook*』
- 『SunOS リファレンスマニュアル』
- 『プログラミングユーティリティ』
- 『*System Services*』
- 『*X/Open CAE Specification*』
- 『*System Interfaces and Headers, Issue4, Release2*』
- 『*Commands and Utilities, Issue4, Release2*』
- 『*System Interface Definitions, Issue4, Release2*』
- 『UI-OSF 日本語環境実装規約 Ver. 1.1』

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 <i>filename</i> と入力します。
『 』	参照する書名を示します。	『コードマネージャ・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 5 章「衝突の回避」を参照してください。

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。

概要

この章では、本マニュアルを読み進むに当たって必要、または知っているとな便利な概念について説明し、関連マニュアルを紹介します。

国際化プログラミング

国際化とは、単一のソフトウェアを最小限のコストで各地域向けに適応させるように設計・実装する手法をいいます。国際化されたプログラムは、特定の国家・地域に依存しない共通の枠組みだけで実装され、次項で説明する「言語対応化」と呼ばれる作業を行うことで、容易に各地域に適合したソフトウェア製品となります。言語対応化とは、国際化されたプログラムに対し特定地域向けに固有の言語情報や文化情報を付加する作業をいいます。

『JFP ユーザーズガイド』の第 1 章では、Solaris における言語対応化の一つとして Japanese Feature Package (以降「JFP」と記述) がどのような日本語化機能を提供しているかを紹介していますので、併せて参照してください。

言語対応化と環境変数

言語対応化したソフトウェアを使用する場合、ユーザーは環境変数を設定します。これにより、ソフトウェアはメッセージや日付、時刻といった情報を各地域・各文化に適合させて動作します。環境変数を用いることでユーザーはソフトウェアの動

作形式を制御できます。どのような環境変数が存在し、どのような制御ができるかについては `setlocale(3C)` および `environ(5)` のマニュアルページを参照してください。

標準への準拠

さまざまな標準化機関が国際化ソフトウェア開発のための仕様・ガイドラインを提供しています。本マニュアルで説明する実例は、基本的に X/Open で規定される CAE 仕様に準拠しています。なお、主として従来の Solaris リリースとの互換性を保つために、実際の動作において XPG で規定された仕様に、厳密には従わない可能性があります。XPG での仕様に厳密に準拠した動作を行うアプリケーションを作成する場合は、`standards(5)` マニュアルページに記載されている手順にしたがってコンパイルおよびリンクを行なってください。また、各リリース毎に対応する XPG のバージョンに関する `standards(5)` マニュアルペーを参照してください。

文字集合とコードセット

言語対応化を行う際に、その地域で使用される文字集合を文字コードとして表現できるようにすることは重要です。しかし、日本などアジア系の地域では使用される文字の数が多いため、すべての文字を表現するには、文字コードを格納するために複数バイトが必要です。Solaris 2.5 までは、Extended UNIX Code (以降「EUC」と記述) と呼ばれるコード体系を採用してこの問題を解決しています。EUC では最大 4 種の文字集合を取り扱うことができ、そのうちの 3 種までは複数バイトの文字集合を割り当てることができます。

『JFP ユーザーズガイド』の第 2 章では、ja ロケールにおける日本語 EUC の定義および文字集合について紹介しています。

CSI

国際化のための基となる文字コード体系に EUC を採用した場合、EUC とは異なる文字コード体系を用いて言語対応化することは原則としてできません。したがっ

て、Solaris 2.5.1 より、システムから EUC に依存する部分を取り除き、X/Open での仕様を基にした特定の文字コード体系に依存しない機構 (これを Code Set Independence と呼び、以降(「CSI」と記述)を導入することで解決しました。CSI の下では、EUC も文字コード体系の 1 種として扱われます。

JFP では、CSI を利用することで Solaris 2.5.1 より「PCK」を、Solaris 7 より UTP-8 をサポートしています。

『JFP ユーザーズガイド』の第 2 章では、ja_JP.PCK ロケールにおける PCK、ja_JP.UTF-8 ロケールにおける UTP-8 の定義および文字集合について紹介しています。

日本語化プログラミングの移植性と利便性

Solaris では大別して 2 種類の日本語ロケール情報を提供しています。1 つは国際化されたプログラムを支援する言語対応化の一つである日本語固有の言語情報と文化情報です。これらは標準化された国際化プログラミングの枠組みを介してアクセスすることができます。この枠組みに則って処理を行う限り、アプリケーションは高い移植性を保ったまま日本語化が行えます。

一方、Solaris で提供する日本語ロケール情報には、国際化の枠組みの中で日本語化を補う付加的な情報も含まれます。これらを利用すると日本語に固有な文字分類や文字対応などの処理が簡単に行える反面、同様の情報を持たない Solaris 以外の環境においてアプリケーションが実行できなくなる可能性があります。こうした付加的な情報を利用するインタフェースについては、第 3 章の中の「日本語 Solaris で拡張されたインタフェース名」で紹介しています。

国際化 API での日本語処理

この章では、XPG で仕様が決定された国際化文字列処理プログラミングインタフェースを紹介し、日本語文字列を処理する方法を説明します。

概説

Solaris 日本語環境は、Solaris に JFP と呼ばれる、日本語に固有の情報を付加したものです。基になる Solaris が XPG に適合しているため、XPG で規定されているインタフェースはすべて利用できます。XPG では国際化プログラミングに必要な API を豊富に用意しており、Solaris でもこれらの API を最大限に活用して日本語処理を行うアプリケーションを開発することができます。

アプリケーションプログラマは、適切なロケール設定の下でこれらの API を通して文字列を処理し、さらにプログラム中に直接日本語文字列を埋め込まない限り、日本語ロケール間の違いを意識せずにアプリケーションを開発することができます。この章では、XPG に規定されている文字列操作の主な API を、複数バイト表現とワイド文字表現向けにそれぞれ表形式でまとめ、実際にこれらを使用したプログラム例を紹介します。

各 API の詳細な仕様については、該当するマニュアルページを参照してください。

複数バイト表現での文字列処理

表 2-1 で主な複数バイト表現の文字列操作のための国際化 API を紹介します。他にも API が用意されています。詳しくはマニュアルページ (Intro (3) など) を参照してください。

表 2-1 主な複数バイト文字列操作 API

インタフェース名	作用
<code>strcat(s1, s2)</code>	s2 を s1 に追加する。追加後の s1 が返る
<code>strncat(s1, s2, n)</code>	s2 のうち最大 n バイトを s1 に追加する。追加後の s1 が返る
<code>strcmp(s1, s2)</code>	s1 と s2 の大小関係を調べる。(順序情報に基づかない)
<code>strncmp(s1, s2, n)</code>	s1 と s2 の最大 n バイトの大小関係を調べる。(順序情報に基づかない)
<code>strcoll(s1, s2)</code>	順序情報に基づき s1 と s2 の大小関係を調べる
<code>strcpy(s1, s2)</code>	s2 を s1 にコピーする。コピー後の s1 が返る
<code>strncpy(s1, s2, n)</code>	s2 の最大 n バイトを s1 にコピーする。コピー後の s1 が返る
<code>strlen(s)</code>	s の長さをバイト数で返す
<code>strxfrm(s1, s2, n)</code>	大小関係を調べるための文字列の変形

注 - このほか文字列に含まれる文字を検出する `strchr()` および `strrchr()` がありますが、複数バイト文字列に対してこれらの API は正しく動作しません。複数バイト文字列に含まれる文字を検出する場合は、文字列を `mbstowcs()` (後述) などでワイド文字列に変換してから `wcschr()` または `wcsrchr()` を使用してください。

プログラム例

複数バイト文字列操作の API を用いたプログラム例を紹介します。使用する場合には `string.h` ヘッダファイルを取り込むこと、また `setlocale()` を処理の最初の段階で呼び出して動作ロケールを適切に設定することが必要です。

この例では文字列を比較しています。一般に文字列の順序関係は文字列を構成する各文字の順序関係で決まり、文字の順序関係はロケールごとに `LC_COLLATE` カテゴリに定義されています。`strcmp()` は比較文字列をバイト単位で比較するため、正しい順序関係を構成しない場合があります。順序情報に基づく比較を行うには `strcoll()` を使用する必要がありますが、一般には `strcmp()` に比べて低速です。`strxfrm()` は、本来 `strcoll()` で比較すべき文字列を変形させます。変形は、変形されたあとの文字列同士を `strcmp()` で比較した結果が変形する前の文字列同士を `strcoll()` で比較した結果と同一になるように行なわれます。データベースを管理するなど多くのデータを順序関係に基づいて並べ換える場合には、効率化が期待できます。

コード例 2-1 において、`main()` 関数から `my_strcoll()` を呼び出していますが、システムが提供する `strcoll()` を呼び出すように変更しても全く同じ結果が得られます。

コード例 2-1 複数バイト文字列操作 API

```
sun% cat my_strcoll.c
/*
 * Read lines from two files, and return the
 * order that is the same as they are compared
 * by strcoll().
 * Comparing will stop if either file reaches EOF.
 * It is assumed that each line has at most BUFSIZ - 1
 * byte length.
 *
 * Actual processing is done by my_strcoll(), which
 * does the followings.
 * 1. Call strxfrm() to get the size of
 * transformed string.
 * 2. Dynamically allocate the memory the
 * buffer. It will be big enough to contain
 * the transformed string and terminating NULL.
 * 3. Call strxfrm() again to get
 * the transformed string. To verify if
 * the error happens, it must clear 'errno'
 * then call strxfrm(). After that, check
 * the value of 'errno.'
 * 4. Call strcmp() with the transformed strings.
 * Since these strings are artificial created,
 * they are not allowed to display.
 */
```

```

#include <stdio.h>
#include <locale.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

static int my_strcoll(const char *, const char *);

int
main(int argc, char *argv[])
{
    FILE *fp1, *fp2;
    char buf1[BUFSIZ], buf2[BUFSIZ];
    char *cp1, *cp2;
    int retval;

    setlocale(LC_ALL, "");

    if (argc != 3) {
        fprintf(stderr, "\tUsage: %s file1 file2\n", argv[0]);
        exit(-1);
    }
    fp1 = fopen((const char *)argv[1], "r");
    fp2 = fopen((const char *)argv[2], "r");
    if ((fp1 == (FILE *)NULL) || (fp2 == (FILE *)NULL)) {
        fprintf(stderr, "%s: Couldn't open %s\n",
            argv[0],
            ((fp1 == (FILE *)NULL) ? argv[1] : argv[2]));
        exit(-1);
    }
    for (;;) {
        cp1 = fgets(buf1, BUFSIZ, fp1);
        cp2 = fgets(buf2, BUFSIZ, fp2);
        if (!cp1 && !cp2) {
            exit(0);
        } else if (!cp1 || !cp2) {
            fprintf(stderr, "%s: No more contents in %s\n",
                argv[0], (cp1 ? argv[2] : argv[1]));
            exit(0);
        }
        retval = my_strcoll((const char *)buf1, (const char *)buf2);
        if (retval == 0) {
            fprintf(stdout, "The same collation order.\n");
        } else if (retval > 0) {
            fprintf(stdout,
                "%s is bigger than %s in terms of collation order.\n",
                argv[1], argv[2]);
        } else {
            fprintf(stdout,
                "%s is bigger than %s in terms of collation order.\n",
                argv[2], argv[1]);
        }
    }
    return (0);
}

static int
my_strcoll(const char *cp1, const char *cp2)
{

```

```

char *transform_1, *transform_2;
size_t xfrm_len1, xfrm_len2;
int ret_coll;

xfrm_len1 = strxfrm((char *)NULL, cp1, (size_t)0);
xfrm_len2 = strxfrm((char *)NULL, cp2, (size_t)0);
transform_1 = (char *)malloc(xfrm_len1 + 1);
transform_2 = (char *)malloc(xfrm_len2 + 1);

errno = 0;
strxfrm(transform_1, cp1, (xfrm_len1 + 1));
if (errno != 0) {
    perror("`my_strcoll(): Error in transforming 1st string'");
    exit(-1);
}
strxfrm(transform_2, cp2, (xfrm_len2 + 1));
if (errno != 0) {
    perror("`my_strcoll(): Error in transforming 2nd string'");
    exit(-1);
}
ret_coll = strcmp((const char *)transform_1, (const char *)transform_2);

free(transform_1);
free(transform_2);

return (ret_coll);
}
sun% cat file 1
入力サンプル 1 です。
This line is identical.
短いです。
sun% cat file 2
入力サンプル 2 です。
This line is identical.
こちらの行は長くなっています。
sun% cc -o my_strcoll my_strcoll.c
sun% ./my_strcoll file1 file2./my_strcoll file1 file2
file2 is bigger than file1 in terms of collation order.
The same collation order.
file1 is bigger than file2 in terms of collation order.
./my_strcoll: No more contents in file1

```

ワイド文字表現での文字列処理

ワイド文字表現の文字列を操作するための主な国際化 API を表 2-2 に紹介します。その他にも API が用意されています。詳しくはマニュアルページ (Intro(3) など) を参照してください。

表 2-2 主なワイド文字列操作 API

インタフェース名	作用
<code>wcscat(ws1, ws2)</code>	<code>ws2</code> を <code>ws1</code> に追加する。追加後の <code>ws1</code> が返る
<code>wcsncat(ws1, ws2, n)</code>	<code>ws2</code> のうち最大 <code>n</code> ワイド文字を <code>ws1</code> に追加する。追加後の <code>ws1</code> が返る
<code>wcschr(ws, wc)</code>	<code>ws</code> の先頭から調べ、最初に出現する <code>wc</code> の位置が返る
<code>wcsrchr(ws, wc)</code>	<code>ws</code> の先頭から調べ、最後に出現する <code>wc</code> の位置が返る
<code>wcscmp(ws1, ws2)</code>	<code>ws1</code> と <code>ws2</code> の大小関係を調べる (順序情報に基づかない)
<code>wcsncmp(ws1, ws2, n)</code>	<code>ws1</code> と <code>ws2</code> の最大 <code>n</code> ワイド文字の大小関係を調べる (順序情報に基づかない)
<code>wcscoll(ws1, ws2)</code>	順序情報に基づき <code>ws1</code> と <code>ws2</code> の大小関係を調べる
<code>wcscpy(ws1, ws2)</code>	<code>ws2</code> を <code>ws1</code> にコピーする。コピー後の <code>ws1</code> が返る
<code>wcsncpy(ws1, ws2, n)</code>	<code>ws2</code> の最大 <code>n</code> ワイド文字を <code>ws1</code> にコピーする。コピー後の <code>ws1</code> が返る
<code>wcslen(ws)</code>	<code>ws</code> の長さをワイド文字数で返す
<code>wcsxfrm(ws1, ws2, n)</code>	大小関係を調べるための文字列の変形
<code>wcwidth(wc)</code>	<code>wc</code> の表示に必要なカラム数
<code>wcswidth(ws, n)</code>	<code>ws</code> 中の最大 <code>n</code> ワイド文字の表示に必要なカラム数

プログラム例

ワイド文字列操作の API を用いたプログラム例を 2 つ紹介します。使用する場合には `wchar.h` ヘッダファイルを取り込むこと、また `setlocale()` を処理の最初の段階で呼び出して動作ロケールを適切に設定することが必要です。

コード例 2-2 はワイド文字列の比較です。コード例 2-1 で複数バイト表現の例として `strxfrm()` と `strcmp()` で `strcoll()` を簡単にシミュレートしましたが、ここではワイド文字表現の場合でのプログラミング例を紹介します。

コード例 2-2 において、`main()` 関数から `my_wcscoll()` を呼び出していますが、システムが提供する `wcscoll()` を呼び出すように変更しても全く同じ結果が得られます。

コード例 2-2 ワイド文字列の比較

```
sun% cat my_wcscoll.c
/*
 * Read lines from two files, and return the
 * order that is the same as they are compared
 * by wcscoll().
 * Comparing will stop if either file reaches EOF.
 * It is assumed that each line has at most BUFSIZ - 1
 * wide char length.
 *
 * Actual processing is done by my_wcscoll(), which
 * does the followings.
 * 1. Call wcsxfrm() to get the size of
 * transformed wide string.
 * 2. Dynamically allocate the memory the
 * buffer. It will be big enough to contain
 * the transformed wide string and terminating (wchar_t)NULL.
 * 3. Call wcsxfrm() again to get
 * the transformed wide string. To verify if
 * the error happens, it must clear 'errno'
 * then call wcsxfrm(). After that, check
 * the value of 'errno.'
 * 4. Call wcscmp() with the transformed wide strings.
 * Since these strings are artificially created,
 * they are not allowed to display.
 */

#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include <errno.h>

static int my_wcscoll(const wchar_t *, const wchar_t *);

int
main(int argc, char *argv[])
{
    FILE *fp1, *fp2;
    wchar_t buf1[BUFSIZ], buf2[BUFSIZ];
    wchar_t*wcp1, *wcp2;
    int retval;

    setlocale(LC_ALL, ``);
    if (argc != 3) {
        fprintf(stderr, ``\tUsage: %s file1 file2\n``, argv[0]);
        exit(-1);
    }
}
```

```

fp1 = fopen((const char *)argv[1], ``r'');
fp2 = fopen((const char *)argv[2], ``r'');
if ((fp1 == (FILE *)NULL) || (fp2 == (FILE *)NULL)) {
    fprintf(stderr, ``%s: Couldn't open %s \n'',
        argv[0],
        ((fp1 == (FILE *)NULL) ? argv[1] : argv[2]));
    exit(-1);
}
for (;;) {
    wcp1 = fgets(buf1, BUFSIZ, fp1);
    wcp2 = fgets(buf2, BUFSIZ, fp2);
    if ((wcp1 == (wchar_t)NULL) && (wcp2 == (wchar_t)NULL)) {
        exit(0);
    } else if ((wcp1 == (wchar_t)NULL) || (wcp2 == (wchar_t)NULL)) {
        fprintf(stderr, ``%s: No more contents in %s\n'',
            argv[0], (wcp1 ? argv[2] : argv[1]));
        exit(0);
    }
    retval = my_wscoll((const wchar_t *)buf1, (const wchar_t *)buf2);
    if (retval == 0) {
        fprintf(stdout, ``The same collation order.\n'');
    } else if (retval > 0) {
        fprintf(stdout,
            ``%s is bigger than %s in terms of collation order.\n'',
            argv[1], argv[2]);
    } else {
        fprintf(stdout,
            ``%s is bigger than %s in terms of collation order.\n'',
            argv[2], argv[1]);
    }
}
return (0);
}

static int
my_wscoll(const wchar_t *wcp1, const wchar_t *wcp2)
{
    xfrm_len1 = wcsxfrm((wchar_t *)NULL, wcp1, (size_t)0);
    xfrm_len2 = wcsxfrm((wchar_t *)NULL, wcp2, (size_t)0);
    transform_1 = (wchar_t *)malloc((xfrm_len1 + 1) * sizeof(wchar_t));
    transform_2 = (wchar_t *)malloc((xfrm_len2 + 1) * sizeof(wchar_t));

    errno = 0;

    wcsxfrm(transform_1, wcp1, (xfrm_len1 + 1));
    if (errno != 0) {
        perror(``my_wscoll(): Error in transforming 1st string'');
        exit(-1);
    }
    wcsxfrm(transform_2, wcp2, (xfrm_len2 + 1));
    if (errno != 0) {
        perror(``my_wscoll(): Error in transforming 2nd string'');
        exit(-1);
    }
    ret_coll =
        wcscmp((const wchar_t *)transform_1, (const wchar_t *)transform_2);

    free(transform_1);
    free(transform_2);

    return (ret_coll);
}

```

```

}
sun% cat file1
入力サンプル 1 です。
This line is identical.
短いです。
sun% cat file2
入力サンプル 2 です。
This line is identical.
こちらの行は長くなっています。
sun% cc -o my_wcscoll my_wcscoll.c
sun% ./my_wcscoll file1 file2
file2 is bigger than file1 in terms of collation order.
The same collation order.
file1 is bigger than file2 in terms of collation order.
./my_wcscoll: No more contents in file1

```

コード例 2-3 は 2 つのファイルを読み込んで、2 段組にして表示する例です。表示の際には 1 行あたり 80 カラムを最大とし、入力ファイルの行末を均等に削除します。複数バイト表現の途中で文字列が分断されないように調整するために、ワイド文字列表現にして処理します。

コード例 2-3 ワイド文字列の比較 (2 段組表示)

```

sun% cat my_pr.c
/*
 * Read lines from two files and merge them into
 * one line so that it doesn't exceed 80 columns.
 * When either file reaches EOF, empty lines will be
 * shown.
 * It is assumed each line has at most BUFSIZ - 1 byte chars
 * and, ASCII space (' ') and vertical bar ('|') character have
 * one display column width.
 * Open files and read lines as wide strings by fgetws().
 * Then call my_pr().
 */

#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <stdlib.h>

#define OUTBUFSIZ 80
#define LIMIT ((OUTBUFSIZ - 3) / 2)
#define WSPC_STR L' '
#define WSPC_CHR L' '
#define WSEPARATOR L'| '
#define WTAB L'\t'
#define TABS 8

static void my_pr(wchar_t *, const wchar_t *, const wchar_t *);
static void put_spc(wchar_t *, int *);

int
main(int argc, char *argv[])
{
    FILE *fp1, *fp2;

```

```

wchar_t buf1[BUFSIZ], buf2[BUFSIZ];
wchar_t outbuf[(OUTBUFSIZ + 1)] = { 0x0 };
int retval;

setlocale(LC_ALL, ``);

if (argc != 3) {
    fprintf(stderr, ``\tUsage: %s file1 file2\n``, argv[0]);
    exit(-1);
}
fp1 = fopen((const char *)argv[1], ``r``);
fp2 = fopen((const char *)argv[2], ``r``);
if ((fp1 == (FILE *)NULL) || (fp2 == (FILE *)NULL)) {
    fprintf(stderr, ``%s: Couldn't open %s \n``,
        argv[0],
        ((fp1 == (FILE *)NULL) ? argv[1] : argv[2]));
    exit(-1);
}
for (;;) {
    if (fgetws(buf1, BUFSIZ, fp1) == (wchar_t *)NULL) {
        buf1[0] = ``0``;
    }
    if (fgetws(buf2, BUFSIZ, fp2) == (wchar_t *)NULL) {
        buf2[0] = ``0``;
    }
    if ((buf1[0] == ``0``) && (buf2[0] == ``0``)) {
        return(0);
    }
    my_pr(outbuf, (const wchar_t *)buf1, (const wchar_t *)buf2);
    fprintf(stdout, ``%S\n``, outbuf);
    outbuf[0] = ``0``;
}

return(0);
}

static void
put_spc(wchar_t *outbufp, int *total)
{
    wcsncat(outbufp, WSPC_STR, 1);
    *total += wwidth(WSPC_CHR);
}

static void
my_pr(wchar_t *outbufp, const wchar_t *wcp1, const wchar_t *wcp2)
{
    int collen;
    int subtotal1, subtotal2;

    for (subtotal1 = 0;
        (collen = wwidth(*wcp1)) <= (LIMIT - subtotal1); wcp1++) {
        if (collen == -1) {
            if (*wcp1 == WTAB) {
                while ((subtotal1 < LIMIT) &&
                    (subtotal1 % TABS) != 1) {
                    put_spc(outbufp, &subtotal1);
                }
            } else {
                continue;
            }
        }
    }
}

```

```

    } else if (collen == 0) {
        break;
    } else {
        wcsncat(outbufp, wcp1, 1);
        subtotal1 += collen;
    }
}

while (subtotal1 < LIMIT) {
    put_spc(outbufp, &subtotal1);
}
wcscat (outbufp, WSEPARATOR);
subtotal1 += wcswidth(WSEPARATOR, wcslen(WSEPARATOR));

for (subtotal2 = 0;
    (collen = wcswidth(*wcp2)) <= (LIMIT - subtotal2); wcp2++) {
    if (collen == -1) {
        if (*wcp2 == WTAB) {
            while ((subtotal2 < LIMIT) &&
                (subtotal2 % TABS) != 1) {
                put_spc(outbufp, &subtotal2);
            }
        } else {
            continue;
        }
    } else if (collen == 0) {
        outbufp[(subtotal1 + subtotal2)] = (wchar_t)NULL;
        break;
    } else {
        wcsncat(outbufp, wcp2, 1);
        subtotal2 += collen;
    }
}

return;
}
sun% cat file
0123456789012345678901234567890123456789012345678901234567890123456789
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
アイウエオカキクケコアイウエオカキクケコアイウエオカキクケコアイウエオカキクケコ
アイウエオカキクケコアイウエオカキクケコアイウエオカキクケコアイウエオカキクケコ
010ア010ア010ア010ア010ア010ア010ア010ア010ア010ア010ア010ア010ア010ア010ア
ア010ア010ア010ア
sun% cat file1
入力サンプル 1 です。
This line is identical.
短いです。
sun% cc -o my_pr my_pr.c
sun% ./my_pr file file1
01234567890123456789012345678901234567 | 入力サンプル 1 です。
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 | This line is identical.
アイウエオカキクケコアイウエオカキクケコアイウエオカキクケコアイウエオカキク | 短いです。
010ア010ア010ア010ア010ア010ア010ア01 |

```


日本語ロケールと文字分類

この章では、XPG において仕様が決定された国際化文字対応・分類プログラミングインタフェース、および Solaris で仕様が追加された日本語ロケール用文字対応・分類インタフェースを紹介し、日本語文字の分類について説明します。

概説

XPG では、各ロケールの LC_CTYPE カテゴリに対して一定の文字クラスを定義し、アプリケーションプログラマが各文字を処理する際の分類 (大文字アルファベットかどうかなど) を支援しています。さらに XPG では、大文字・小文字の対応を定義する仕組みも LC_CTYPE の中で定義しているため、`toupper()` や `tolower()` などの API を通して簡単に大文字小文字の変換ができます。

このような XPG が規定する文字クラス分類・文字対応 API には、それぞれバイト単位処理・ワイド文字表現単位処理の両方が用意されています。

Solaris では日本語テキストで使用する文字 (漢字など) を同様の API で処理できるように、日本語ロケール専用の文字分類クラスおよび文字対応を追加しています。これらの文字対応と文字分類の処理は、各文字をワイド文字表現に変換して API を呼び出します。Solaris における JFP で提供された日本語ロケールの定義については『JFP ユーザーズガイド』の第 2 章の表 2-3 「X/Open で規定された文字分類クラス名と日本語ロケールでの定義範囲」を参照してください。

日本語文字対応

文字対応の操作のための API のうち、日本語ロケールの文字集合の処理に有効な API を表 3-1、表 3-2 に示します。その他にも API が用意されています。詳しくはマニュアルページ (`towctrans(3C)`、`wctrans(3C)`、`wctrans_ja(3C)` など) を参照してください。

表 3-1 文字対応 API その 1

XPG で規定されるインタフェース名	作用
<code>towupper(wc)</code>	ワイド文字 <code>wc</code> に対応する大文字ワイド文字表現を返す。なければ <code>wc</code> をそのまま返す
<code>towlower(wc)</code>	ワイド文字 <code>wc</code> に対応する小文字ワイド文字表現を返す。なければ <code>wc</code> をそのまま返す
<code>towctrans(wc, wctrans("タグ名"))</code>	タグ名に基づいてワイド文字 <code>wc</code> に対応したワイド文字表現を返す。なければ <code>wc</code> をそのまま返す
<code>wctrans("タグ名")</code>	<code>towctrans()</code> で使う、タグ名に対応する値を返す。タグ名として以下が使用できる。 <code>"tolower", "toupper"</code>

表 3-2 文字分類 API その 2

日本語 Solaris で拡張されたインタフェース名	作用
<code>wctrans("タグ名")</code>	<code>towctrans()</code> で使う、タグ名に対応する値を返す。 Solaris では日本語ロケールにおいて以下をタグ名として使用できる <code>"tojhira" "tojkata" "tojisx0208"</code> <code>"tojisx0201"</code>

プログラム例

日本語文字対応の API を用いたプログラム例を 2 つ紹介します。使用する場合には `wchar.h`、`wctype.h` ヘッダファイルを取り込むこと、また `setlocale()` を処理の最初の段階で呼び出して動作ロケールを適切に設定することが必要です。

コード例 3-1 では、入力ファイル中に存在する JIS X 0201 かな文字 (半角カナ) を JIS X 0208 かな文字に変換するフィルタを紹介します。既存のアプリケーション・ネットワークの中には、規約上、または実装上の制限により JIS X 0201 かな文字を使用したデータ通信ができないものがあります。そのような環境に対しては通信に先立ち、JIS X 0201 かな文字を JIS X 0208 かな文字に変換するなど入力ファイルの加工が必要です。この例では、入力ファイルを 1 行ずつワイド文字列として読み込み、各ワイド文字を `towctrans()` で変換しています。

コード例 3-1 日本語文字対応

```
sun% cat my_kanato208.c

/*
 * Read lines from a file and convert JIS X 0201 kana
 * characters to the correspondent ones in JIS X 0208
 * set. This will stop processing if the input file
 * reaches EOF. It is assumed that each line has
 * at most BUFSIZ -1 wide char length.
 *
 * Actual processing is done by my_kanato208(), which
 * does the followings.
 * 1. Get the length of wide string.
 * 2. Convert each wide char from the top
 * of the string by applying towctrans().
 * (The return value of towctrans() will be
 * the same if there's no correspondent char.)
 * 3. Write the correspondent wide char to
 * original string.
 */
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>

static void my_kanato208(wchar_t *);

int
main(int argc, char *argv[])
{
    wchar_t buf[BUFSIZ];

    setlocale(LC_ALL, "");

    while (fgetws(buf, BUFSIZ, stdin) != (wchar_t *)NULL) {
        my_kanato208(buf);
        fprintf(stdout, "%S", buf);
    }
    return (0);
}
```

```

}

static void
my_kanato208(wchar_t *wcp)
{
    size_t wstr_len;
    wint_t retval;
    int index;

    wstr_len = wcslen(wcp);
    for (index = 0; index < wstr_len; index++) {
        retval = towctrans((wint_t)wcp[index],
wctrans("tojisx0208"));
        wcp[index] = retval;
    }
}

```

```
sun% cat file3
```

新しいシステム*は現在のネットワーク環境を変えることなく
インターネット*とのシームレス*な接続を可能にします。また
セキュリティ*の問題も新しい認証テクノロジー*を用いることで
アドミニストレータ*の負担を減らしています。

```
sun% cc -o my_kanato208 my_kanato208.c
```

```
sun% cat file3 | ./my_kanato208
```

新しいシステム*は現在のネットワーク*環境を変えることなく
インターネット*とのシームレス*な接続を可能にします。また
セキュリティ*の問題も新しい認証テクノロジー*を用いることで
アドミニストレータ*の負担を減らしています。



注意 - * の部分のカタカナは、半角カタカナになります。

コード例 3-2 は `wcstol()` の拡張例です。現在の Solaris が提供する日本語ロケールでは、JIS X 0208 文字集合で表された数値文字列に対して、直接 `wcstol()` を呼び出すことができません。そこで、数値文字列をワイド文字列データとして読み込み、`towctrans()` で対応する JIS X 0201 文字に変換し、`wcstol()` を呼び出しています。

コード例 3-2 `wcstol()` の拡張

```
sun% cat my_wcstol.c
```

```

/*
 * Read lines from a file and convert tokenized
 * wide char string to long integer.
 * Conversion will stop if the input file reaches
 * EOF, and output the sum of input integers.
 * It is assumed that each line has at most
 * BUFSIZ - 1 wide char length.
 *
 * Actual conversion is done by my_wcstol(), which
 * does the followings.
 * 1. Get the length of wide char string.
 * 2. Convert each wide char from the top
 * of the string by applying towctrans().

```

```

*           The correspondent JIS X 0201 wide char value
*           will be gotten for each JIS X 0208 digit chars
*           in the string.
*           (The return value of towctrans() will be
*           the same if there's no correspondent char.)
*   3.       Write the correspondent wide char to
*           original string.
*   4.       Call wcstol() with the converted wide string.
*/
#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include <errno.h>

#define      WRET          L'\n'

static long my_wcstol(wchar_t *, wchar_t **, int);

int
main(int argc, char *argv[])
{
    wchar_t buf[BUFSIZ];
    wchar_t *headp, *nextp;
    long retval, total;
    setlocale(LC_ALL, "");
    total = retval = 0;
    while (fgetws(buf, BUFSIZ, stdin) != (wchar_t *)NULL) {
        headp = buf;
        while (headp != (wchar_t *)NULL) {
            errno = 0;
            retval = my_wcstol(headp, 0);
            if (errno != 0) {
                if (nextp[0] == WRET) {
                    break;
                } else {
                    perror("my_wcstol()");
                    exit (-1);
                }
            }
            fprintf(stdout, "retval = [%ld]\n", retval);
            total += retval;
            headp = nextp;
        }
        fprintf(stdout, "Total = %ld.\n", total);
        return (0);
    }
}

static long
my_wcstol(wchar_t *wcp, wchar_t **endp, int base)
{
    size_t wstr_len;
    wint_t retval;
    int index;
    long ret_val;
    wstr_len = wcslen(wcp);
    for (index = 0; index < wstr_len; index++) {
        retval = towctrans((wint_t)wcp[index], wctrans("tojisx0201"));
        wcp[index] = (wchar_t)retval;
    }
    ret_val = wcstol((const wchar_t *)wcp, endp, base);
}

```

```

        return (ret_val);
    }
sun% cat file4
343 34534
3 4 5 3 4 5
3 9 8 5 7      3 9 8
sun% cc -o my_wcstol my_wcstol.c
sun% ./my_wcstol < file4
retval = [343]
retval = [34534]
retval = [12]
retval = [345345]
retval = [345345]
retval = [39857]
retval = [398]
retval = [5834589]
Total = 6600423.

```

日本語文字分類

文字分類の操作のための API のうち、日本語ロケールの文字集合の処理に有効な API を表 3-3、表 3-4 に示します。その他にも API が用意されています。詳しくはマニュアルページ (`iswalph(3C)`、`wctype(3C)`、`wctype_ja(3C)`)などを参照してください。

表 3-3 文字分類 API その 1

XPG で規定されるインタフェース名	作用
<code>iswctype(wc, type)</code>	<code>wc</code> が <code>type</code> クラスに属するかどうか調べる
<code>wctype("タイプ名")</code>	<p><code>iswctype()</code> の第 2 引数を、タイプ名から作成する。XPG で標準文字クラスとして規定されているものは以下のとおり</p> <pre> "alnum" "alpha" "cntrl" "digit" "graph" "lower" "print" "punct" "space" "upper" "xdigit" "blank" </pre>

表 3-4 文字分類 API その 2

日本語 Solaris で拡張されたインタフェース名	作用
wctype("タイプ")	<p>iswctype() の第 2 引数を、タイプ名から作成する。Solaris で日本語ロケール向けに拡張された文字クラスは以下のとおり</p> <pre> "jkanji"() "jkata"() "jhira"() "jdigit"() "jparen"() "jline"() "jisx0201r"() "jisx0208"() "jisx0212"() "udc"() "vdc"() "jalpha" "jspecial" "jgreek" "jrussian" "junit" "jsci" "jgen" "jpunct" </pre>

プログラム例

日本語文字分類の API を用いたプログラム例を紹介します。前述の文字対応 API の場合と同様に、使用する場合には `wchar.h` ヘッダファイルを取り込むこと、また `setlocale()` を処理の最初の段階で呼び出して動作ロケールを適切に設定することが必要です。

コード例 3-3 では、入力ファイルをワイド文字列として読み込んで入力中の JIS X 0208 ひらがなとカタカナを交換し、JIS X 0208 数字文字は ASCII に変換して出力します。

コード例 3-3 文字分類

```

sun% cat my_charconv.c
/*
 * Read lines from a file and convert JIS X 0208 hiragana
 * characters to JIS X 0208 katakana characters, and
 * vice versa. In addition, JIS X 0208 digit characters
 * are converted to the correspondent ones in JIS X 0201
 * characters.
 * Conversion will stop if the input file reaches EOF.
 * It is assumed that each line has at most BUFSIZ - 1
 * wide char length.
 *
 * Actual conversion is done by my_charconv(), which does
 * the followings.

```

```

* 1. Get the length of the wide string.
* 2. Convert each wide char from the top
*   of the string by applying towctrans().
*   (The return value of towctrans() will be
*    the same if there's no correspondent char.)
* 3. Write the correspondent wide char to
*   original string and output it.
*/

#include <stdio.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include <jctype.h>
#include <errno.h>

#define WRET L'\n'

static int my_charconv(wchar_t *);

int
main(int argc, char *argv[])
{
    wchar_t buf[BUFSIZ];
    wchar_t *headp, *nextp;
    long retval, total;

    setlocale(LC_ALL, '');
    total = retval = 0;

    while (fgetws(buf, BUFSIZ, stdin) != (wchar_t *)NULL) {
        retval = my_charconv(buf);
        if (retval == -1) {
            perror('my_charconv()');
            exit(-1);
        }
        fprintf(stdout, '%S', buf);
    }

    return (0);
}

static int
my_charconv(wchar_t *wcp)
{
    size_t wstr_len;
    wint_t retval;
    int index;
    long ret_val;

    wstr_len = wcslen(wcp);
    for (index = 0; index < wstr_len; index++) {
        errno = 0;
        if (iswctype((wint_t)wcp[index], wctype('jhira')))
            retval = towctrans((wint_t)wcp[index], wctrans('tojkata'));
        else if (iswctype((wint_t)wcp[index], wctype('jkata')))
            retval = towctrans((wint_t)wcp[index], wctrans('tojhira'));
        else if (iswctype((wint_t)wcp[index], wctype('jdigit')))
            retval = towctrans((wint_t)wcp[index], wctrans('tojisx0201'));
        else

```

```
    retval = wcp[index];

    if (errno != 0)
        return (-1);
    wcp[index] = (wchar_t)retval;
}

return (0);
}
sun% cat file5
ひらがなはかたかなに置換されます。
カタカナハヒラガナニ置換サレマス。
漢字、記号、全角a l p h a b e tや
JIS X 0201 カナナドハ* 置換 サレマセン*。
sun% cc -o my_charconv my_charconv.c
sun% ./my_charconv < file5
ヒラガナハカタカナニ置換サレマス。
かたかなはひらがなに置換されます。
漢字、記号、全角a l p h a b e tや
JIS X 0201 カナナドハ* 置換 サレマセン*。
```



注意 - * の部分のカタカナは、半角カタカナになります。

日本語文字コード変換

この章では、文字列の複数バイト表現とワイド文字表現の相互変換のためのインタフェース、および `iconv()` 汎用コード変換インタフェースを紹介し、その利用方法を説明します。

概説

Solaris では、テキストデータを複数バイト表現 (ファイルコードとも呼ぶ) の形でファイルに格納します。一方、アプリケーションの内部でテキストデータを取り扱う際には、第 2 章、第 3 章で紹介したように、ワイド文字表現 (プロセスコードとも呼ぶ) の形で行うと便利な場合があります。したがって、日本語データを扱うアプリケーションでは複数バイト表現とワイド文字表現の相互変換を行う場面がしばしば生じます。XPG ではこの用途の API を用意しており、アプリケーションでのさまざまな相互変換を支援します。

また、日本語テキストで使用される文字集合は同じでもエンコーディングが異なるために、アプリケーションがテキストデータとして正しく取り扱えない場合があります。アプリケーションが自らの責任でエンコーディングの変換を行うように開発することもできますが、その場合、アプリケーションは変換可能なすべてのエンコーディングに関する情報を含んでいなくてはなりません。また、新しいエンコーディングをサポートする場合には、アプリケーションの再コンパイルが必要になります。

このような事態を避けるためには、コード変換に関するルール、または変換サービスそのものをアプリケーション本体から切り離し、変換前のエンコーディングと変

換後のエンコーディングを指定して、動的に変換サービスを呼び出します。XPG ではこの用途の API を用意しており、汎用的なコード変換を支援します。

複数バイト・ワイド文字の相互変換

表 4-1 で主な複数バイト・ワイド文字の相互変換のための API を紹介します。この他、`printf(3S)`、`scanf(3S)` などのマニュアルページも参照してください。

表 4-1 複数バイト・ワイド文字相互変換 API

インタフェース名	作用
<code>mbtowc(pwc, s, n)</code>	<code>s</code> の先頭から最大 <code>n</code> バイト調べ、複数バイト 1 文字分をワイド文字表現にして <code>pwc</code> へ格納
<code>mbstowcs(pwcs, s, n)</code>	<code>s</code> の先頭から複数バイト文字列をワイド文字列に変換する。最大 <code>n</code> ワイド文字変換したら終了
<code>wctomb(s, wc)</code>	ワイド文字 <code>wc</code> を複数バイト表現に変換し <code>s</code> へ格納
<code>wcstombs(s, pwcs, n)</code>	<code>pwcs</code> からワイド文字列を複数バイト表現に変換しながら <code>s</code> に格納。変換した複数バイトの合計が最大 <code>n</code> バイトになれば終了
<code>mblen(s, n)</code>	<code>s</code> の先頭から最大 <code>n</code> バイト調べ複数バイト 1 文字分を構成するバイト数を返す
<code>fgetwc(stream)</code>	入力ストリームから 1 複数バイト分を読み込みワイド文字表現で返す
<code>ungetwc(wc, stream)</code>	ワイド文字 <code>wc</code> を <code>stream</code> へプッシュバックする
<code>fgetws(ws, n, stream)</code>	入力ストリーム <code>stream</code> から複数バイト文字列を読み込み、最大 <code>n-1</code> ワイド文字分を <code>ws</code> に格納する
<code>fputwc(wc, stream)</code>	出力ストリーム <code>stream</code> へワイド文字 <code>wc</code> を出力
<code>fputws(ws, stream)</code>	出力ストリーム <code>stream</code> へワイド文字列 <code>ws</code> を出力

プログラム例

コード例 4-1 では、あるファイルに対してこれらの API を適用した複数バイト・ワイド文字の相互変換のプログラム例を紹介します。これらの API を使用する場合は、適切なヘッダファイルを取り込むこと

(`mbtowl()`、`mbstowcs()`、`wctomb()`、`wcstombs()`、`mblen()` は `stdlib.h` を、`ungetwc()`、`fgetws()`、`fputwc()`、`fputws()` は `wchar.h` を取り込む) および `setlocale()` を処理の最初の段階で呼び出して動作ロケールを適切に設定することが必要です。

コード例 4-1 複数バイト・ワイド文字の相互変換

```
sun% cat my_mbwc.c

/*
 * Read lines from stdin and
 * count the number of chars
 * that belong to specific category.
 * Counting will stop if input reaches
 * EOF. It is assumed that each line
 * has at most BUFSIZ - 1 byte length.
 *
 * To categorize each chars, iswctype()
 * is used. Therefore, it is necessary
 * to convert the input multibyte buffer
 * to the wide char buffer. mbstowcs()
 * is called for that purpose.
 */

#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include <wchar.h>

static char mdbuf[BUFSIZ];
static wchar_t wcbuf[BUFSIZ];

int
main(int argc, char *argv[])
{
    size_t retval;
    int i, alpha_char, ideo_char, kana_char, other_char;

    setlocale(LC_ALL, "");
    alpha_char = ideo_char = kana_char = other_char = 0;
    while(fgets(mdbuf, BUFSIZ, stdin) != NULL) {
        retval = mbstowcs(wcbuf, mdbuf, BUFSIZ);
        if (retval == (size_t)-1) {
            fprintf(stderr, "Invalid char is found during mbstowcs()\n");
            exit(-1);
        }
        retval = wcslen((const wchar_t *)wcbuf);
        for (i = 0; i < retval; i++) {
            if (iswctype(wcbuf[i], wctype("jisx0201r"))) {
            } else if (iswctype(wcbuf[i], wctype("alpha"))) {

```

```

        alpha_char++;
    } else {
        other_char++;
    }
}
printf(stdout, "The input consist of\n");
printf(stdout, "%d Alphabetical chars,\n", alpha_char);
printf(stdout, "%d JIS X 0208 Kanji chars,\n", ideo_char);
printf(stdout, "%d JIS X 0201 Kana chars and\n", kana_char);
printf(stdout, "%d other chars.\n", other_char);
return(0);
}
sun% cc -o my_mbwc my_mbwc.c
sun% cat file6
/* Here's the content of file3 */
新しいシステム*は現在のネットワーク*環境を変えることなく
インターネット*とのシームレス*な接続を可能にします。また
セキュリティ*の問題も新しい認証テクノロジー*を用いることで
アドミニストレータ*の負担を減らしています。
/* Here's the content of file5 */
ひらがなはかたかなに置換されます。
カタカナハヒラガナニ置換サレマス。
漢字、記号、全角 a l p h a b e t や
JIS X 0201 カナナドハ* 置換 サレマセン*。
sun% ./my_mbwc < file6
The input consist of
54 Alphabetical chars,
31 JIS X 0208 Kanji chars,
56 JIS X 0201 Kana chars and
117 other chars.

```



注意 - * の部分のカタカナは、半角カタカナになります。

汎用コード変換

汎用コード変換の API は表 4-2 のとおりです。変換前エンコーディングと変換後エンコーディングを指定する名称と指定の意味については `iconv_ja(5)` のマニュアルページ、および『JFP ユーザーズガイド』の第 6 章「フィルタを用いたコード変換」を参照してください。

表 4-2 汎用コード変換 API

インタフェース名	作用
<code>iconv_open()</code>	変換元コードと変換先コードから変換に必要な情報を得る
<code>iconv()</code>	得られた情報をもとに実際の変換を行う
<code>iconv_open()</code>	変換に必要なだった情報を解放する

プログラム例

コード例 4-2 では `iconv()` インタフェースを用いて `iconv(1)` コマンドのサブセットに相当するフィルタを作成します。これらの API を使用する場合は、`iconv.h` ヘッダファイルを取り込む必要があります。一般的な国際化 API と異なり、`iconv()` では、変換前、変換後のそれぞれの文字集合およびエンコーディングに関する情報を変換記述子を通して入手します。したがって、次のプログラム例でも `setlocale()` が呼び出されていないことに注意してください。

コード例 4-2 汎用コード変換

```
sun% cat my_iconv.c

/*
 * Read lines from a stdin and convert the encoding.
 * It is assumed that each line has at most BUFSIZ - 1
 * byte length.
 * Both of source and destination encodings are passed
 * from the command line.
 *
 * Note:      Calling iconv() itself doesn't need to call
 *            setlocale() in advance.
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <iconv.h>
int
main(int argc, char *argv[])
{
    iconv_t icv_hook;
    char in_buf[BUFSIZ];
    char out_buf[BUFSIZ];
```

```

char *inp;
char *outp;
char *from_code;
char *to_code;
extern char *optarg;
extern int optind;
size_t ret_val;
size_t in_buf_left;
size_t out_buf_left;
int i;

if (argc != 5) {
    fprintf(stderr, "usage: %s -f -t \n", argv[0]);
    exit(-1);
}
while ((i = getopt(argc, argv, "f:t:")) != EOF)
    switch (i) {
        case 'f':
            from_code = optarg;
            break;
        case 't':
            to_code = optarg;
            break;
        default:
            fprintf(stderr, "usage: %s -f -t \n", argv[0]);
            exit(-1);
    }
icv_hook = iconv_open(to_code, from_code);
if (icv_hook == (iconv_t)-1) {
    perror("iconv_open()");
    exit(-1);
}

i = 0;
while(fgets(in_buf, BUFSIZ, stdin) != NULL){
    if (!in_buf[0]) {
        perror("fgets()");
        exit(-1);
    }
    i++;
    memset(out_buf, 0, BUFSIZ);
    in_buf_left = strlen(in_buf);
    out_buf_left = BUFSIZ;
    inp = in_buf;
    outp = out_buf;
    errno = 0;
    ret_val = iconv(icv_hook,
                    (const char **)inp, in_buf_left, outp, out_buf_left);
    if (ret_val == (size_t)-1) {
        if (errno == EILSEQ)
            perror("EILSEQ");
        else if (errno == E2BIG)
            perror("E2BIG");
        else if (errno == EINVAL)
            perror("EINVAL");
        fprintf(stderr, "Line number is %d\n", i);
        exit(-1);
    }
    write(STDOUT_FILENO, out_buf, (BUFSIZ - out_buf_left));
}

```

```
    }
    iconv_close(icv_hook);
    return(0);
}
sun% cat file3
新しいシステム*は現在のネットワーク*環境を変えることなく
インターネット*とのシームレス*な接続を可能にします。また
セキュリティ*の問題も新しい認証テクノロジー*を用いることで
アドミニストレータ*の負担を減らしています。
sun% cc -o my_iconv my_iconv.c
sun% cat file3 | ./my_iconv -f eucJP -t PCK | ./my_iconv -f PCK -t eucJP
新しいシステム*は現在のネットワーク*環境を変えることなく
インターネット*とのシームレス*な接続を可能にします。また
セキュリティ*の問題も新しい認証テクノロジー*を用いることで
アドミニストレータ*の負担を減らしています。
```



注意 - * の部分のカタカナは、半角カタカナになります。

メッセージ処理

この章では、メッセージ処理のためのインタフェースを紹介し、国際化・日本語化されたアプリケーションでの日本語のメッセージを取り扱い、表示させる方法を紹介します。

概説

アプリケーションプログラムは、その使用目的や動作の形態によって、一般ユーザーに対してさまざまな局面でテキストメッセージを利用してフィードバックを行います。エラーメッセージや入力を促すプロンプト文字列、GUI を用いたアプリケーションのボタンラベルや階層化メニューなどがこれに相当します。

このようなアプリケーションを開発する場合、開発者はメッセージとして日本語テキストを使用できますが、ソースコード中に直接日本語テキストを挿入するようなプログラミングはお勧めできません。挿入されたテキストは特定のロケール環境に依存したエンコーディングで表現されるため、結果としてアプリケーション全体がロケールに依存することになります。また、メッセージだけを変更する場合にも、アプリケーション全体を再コンパイルしなくてはなりません。

かわりに、アプリケーションプログラムはメッセージをソースコード本体から切り離し、メッセージファイルとして提供して、アプリケーションが実行時に動作ロケールに応じて動的にメッセージファイルを参照できるような仕組み(以下「メッセージ処理」と記述)を利用できます。これにより、アプリケーションを一度コンパイルすると、メッセージを表示したいロケールの翻訳メッセージファイルを提供することで、容易にアプリケーション中のテキストを翻訳できます。

Solaris では、メッセージ処理の API を 2 種類提供しています。1 つは米国 Sun Microsystems, Inc. 独自のインタフェースである `gettext()` を用いる方式で、もう 1 つは X/Open で規定された `catgets()` を用いる方式です。これらに互換性はありません。アプリケーションの移植性が問題となる場合には、X/Open で規定されている方式を用いてメッセージ処理することをお勧めします。

アプリケーションのメッセージ処理を行う基本的な手順を以下に説明し、その手順に従って X/Open 方式でのメッセージ処理 API を説明します。

アプリケーションのメッセージ処理手順

1. 翻訳したいメッセージに印を付け、そのメッセージが属する集合 (セットまたはテキストドメイン) を指定します。
複数のアプリケーションで同じメッセージが存在する可能性があるためです。
2. 翻訳したいメッセージをソースコードから抽出し、オリジナルメッセージだけのファイルを作ります。
3. 36 ページの手順 2 で作成したファイルに翻訳したメッセージを追加し、アプリケーションが参照できる形式に変換します。
4. 変換したファイルを一定の場所に置き、アプリケーションがパス名とファイル名から翻訳メッセージのあるファイルを特定し、参照できるようにプログラムを変更します。

X/Open 方式

X/Open 方式では、翻訳メッセージは「メッセージカタログ」と呼ばれるファイルに格納されます。メッセージカタログファイルは 1 つ以上のセットに分かれ、それぞれのセットの中には 1 つ以上の翻訳メッセージがメッセージ ID と共に存在します。アプリケーションは、オリジナルのメッセージに対して、メッセージカタログファイル、セット ID、メッセージ ID を指定して翻訳メッセージを使用します。

メッセージ処理の手順は次のとおりです。

1. 翻訳したいメッセージに印を付け、そのメッセージが属する集合を指定します。
印をつける操作は、メッセージ文字列を `catgets(3C)` インタフェースで囲うことで行います。その際に、翻訳メッセージを格納するカタログファイル、セット ID、インデックス ID をあらかじめ決めておきます。`catgets()` では、「メッ

「メッセージカタログ記述子」を介してカタログファイルを利用するため、`catopen(3C)` インタフェースで取得しておきます。

```
nl_catd hook;
hook= catopen(`catalog', 0)
fprintf(stdout, catgets(hook, SET_1, ID_1, `Defaults:'));
```

2. 翻訳したいメッセージをソースコードから抽出し、オリジナルメッセージだけのファイルを作ります。

メッセージカタログファイルの中身を作成します。各メッセージはメッセージ ID を持ち、セット ID と共に管理されます。アプリケーションからは、これらの ID だけで利用されるので、オリジナルメッセージと別の ID にならないように注意が必要です。

```
sun% cat my_message.orig
$set 1 my_message set
$ 1 Here's the 1st message.\n
$ 2 Here's the 2nd message from %s.\n
```

3. 37ページの手順 2 で作成したファイルに翻訳したメッセージを追加し、アプリケーションが参照できる形式に変換します。

カタログファイルの変換は `gencat` コマンドを使用します。`gencat` コマンドが処理できるファイル形式については `gencat(1)` のマニュアルページを参照してください。

```
sun% cp my_message.orig my_message.ja
sun% vi my_message.ja
sun% cat my_message.ja
$set 1 my_message set
$ 1 Here's the 1st message.\n
1 これは第 1 のメッセージです.\n
$ 2 Here's the 2nd message from %s.\n
2 これは %s からの第 2 のメッセージです.\n
sun% gencat my_catalogue.cat my_message.ja
```

4. 変換したファイルを一定の場所に置き、アプリケーションがパス名とファイル名から翻訳メッセージのあるファイルを特定し、参照できるようにプログラムを変更します。

カタログファイルを、36ページの手順 1 で示したように利用するには、`catopen()` でメッセージカタログ記述子を取得する必要があります。不要になった記述子は `catopen()` インタフェースを介して解放できます。

catopen() でカタログファイルを利用するには、カタログファイル名を絶対パス名で指定する方法と、NLSPATH 環境変数およびデフォルトパスを利用した相対パス名で指定する方法があります。詳細は catopen(3C) のマニュアルページを参照してください。

なお、これらの API を使用する場合は、nl_types.h ヘッダファイルを取り込むこと、また setlocale を処理の最初の段階で呼び出して動作ロケールを適切に設定することが必要です。

```
sun% cat my_message.c
#include <stdio.h>
#include <locale.h>
#include <nl_types.h>
#include <errno.h>

#define MY_CATALOGUE ``my_catalogue.cat``
#define SAMPLE_SET 1
#define SAMPLE_MES_IND 1

int
main(int argc, char *argv[])
{
    int     retval;
    nl_catd hook;

    setlocale(LC_ALL, ``);
    hook = catopen(MY_CATALOGUE, 0);
    if (hook == (nl_catd)-1) {
        perror(``catopen()'');
        exit(-1);
    }
    fprintf(stdout, catgets(hook, SAMPLE_SET, SAMPLE_MES_IND,
        ``Here's the 1st message.\n''));
    fprintf(stdout, catgets(hook, SAMPLE_SET, (SAMPLE_MES_IND + 1),
        ``Here's the 2nd message from %s.\n''), argv[0]);
    errno = 0;
    retval = catclose(hook);
    if (retval != 0) {
        perror(``catclose()'');
        exit(-1);
    }

    return (0);
}
sun% mkdir -p /usr/tmp/C /usr/tmp/ja
sun% cp my_catalogue.cat /usr/tmp/ja
sun% setenv NLSPATH /usr/tmp/%L/%N
sun% cc -o my_message my_message.c
sun% env LANG=C ./my_message
Here's the 2nd message from ./my_message.
sun% ./my_message
これは第 1 のメッセージです。
これは ./my_message からの第 2 のメッセージです。
```

シェルスクリプトの国際化と日本語化

この章では、シェルスクリプトに対するロケール環境の設定とメッセージ処理コマンドを紹介し、その利用方法を説明します。

概説

これまでの章では、主にバイナリアプリケーションでの日本語処理について説明してきました。一方で Solaris では XPG に準拠した国際化されたコマンド・ユーティリティが豊富に用意されており、これらを組み合わせることで日本語を扱うシェルスクリプトが容易に作成できます。また、メッセージ処理を行うコマンドが提供され、これらのシェルスクリプト内部のメッセージに対して、ロケールに応じた翻訳メッセージを表示することができます。

国際化に影響するロケール環境変数

スクリプトファイルから各コマンド・ユーティリティを起動して日本語を適切に処理したり、エラーメッセージなどを日本語で表示するには、スクリプトを実行するシェルに対して正しく動作ロケールの設定を行い、その環境の下で各コマンド・ユーティリティが処理できるようにする必要があります。これらも一連のロケール環境変数を設定することで行えます。シェルに対してロケールを設定する方法については、『JFP ユーザーズガイド』の第 3 章、「環境変数によるロケールの設定」を参照してください。

スクリプトのメッセージ処理

シェルスクリプト内部で翻訳メッセージを出力したい場合は、`gettext(1)` コマンドを使用します。メッセージ処理によって翻訳メッセージを用意し、表示させるための手順は次のとおりです。第5章で紹介した `catgets()` を用いる場合とは、翻訳メッセージを格納するファイルの構造が異なりますので注意してください。

1. テキストドメインおよび翻訳メッセージを指定します。

テキストドメインはメッセージが属する集合です。テキストドメインを単位にして、メッセージファイルが作成されます。`echo` や `printf` などで出力すべきメッセージは、`gettext(1)` コマンドを介して取得されます。その際 `gettext` の引数として、メッセージオブジェクトおよびデフォルトのメッセージを指定します。デフォルトメッセージはメッセージオブジェクトを検索する際のインデックスとして用いられるほかに、対応する翻訳メッセージが取得できなかった場合に表示される文字列になります。なお、動作ロケールが C の場合は、`gettext` は常にデフォルトメッセージを表示します。

```
gettext 'my_domain' ``Usage: Test <args> ...\n``
prt_total() {
    str='gettext 'sample' ``Total:``
    printf ``%s %s\n`` ``$str`` $1
}
```

2. 「ポータブルオブジェクト」と呼ばれるテキストファイルを作成します。

ポータブルオブジェクトファイルは複数のテキストドメインをまとめて格納できます。各テキストドメインは `domain "domainname"` という行により区切られます。また各行は `msgid "message_identifier"`、`msgstr "messagestring"` の対で表されます。`msgid` 行に現われたメッセージを翻訳し、`msgstr` 行に記述します。詳しい表記形式については `msgfmt(1)` のマニュアルページを参照してください。これらのファイルは接尾辞 `.po` を付けたファイル名で保存するとわかりやすいでしょう。

```
sun% cat my_message.orig
domain "my_domain"
msgid "Usage: my_script <args> ... \n"
msgstr "使用方法 : my_script 引数 ... \n"
msgid "arg is"
msgstr "引数は"
```

3. ポータブルオブジェクトファイルに msgfmt コマンドを適用し、「メッセージオブジェクト」と呼ばれるバイナリファイルを作成します。

メッセージオブジェクトはロケールに依存します。入力したポータブルオブジェクトファイルが複数のテキストドメインを持つ場合、そのドメインの数だけメッセージオブジェクトファイルが作成されます。これらのファイルはmsgfmt コマンドにより接尾辞 .mo を付けたファイル名を持ちます。msgfmt コマンドの詳細についてはマニュアルページを参照してください。

```
sun% ls my_domain*
my_domain.po
sun% msgfmt my_domain.po
sun% ls my_domain*
my_domain.mo    my_domain.po
```

4. メッセージオブジェクトをインストールします。

アプリケーションスクリプト内部で翻訳メッセージを参照する場合は、40ページの手順1のように gettext コマンドを使用します。その際に TEXTDOMAIN 環境変数を利用して、参照すべきテキストドメインを指定します。テキストドメインファイルは、通常、システムのデフォルトのディレクトリ

(/usr/lib/locale/lang/LC_MESSAGES) 以下を検索しますが、

TEXTDOMAINDIR 環境変数を使用して検索ディレクトリを変更することもできます。詳細については gettext(1) のマニュアルページを参照してください。

```
sun% cat my_script.sh
#!/usr/bin/sh
LANG=${LANG:="C"}
export LANG
TEXTDOMAIN=${TEXTDOMAIN:='my_domain'}
export TEXTDOMAIN

help() {
    gettext ${TEXTDOMAIN} "Usage: my_script ...\n"
}

do_print() {
    str=`gettext ${TEXTDOMAIN} "arg is"`
    printf "%s %s\n" "$str" $1
}

if [ $# -le 0 ]
then
    help
    exit 0
fi

str=`gettext ${TEXTDOMAIN} "arg is"`
while [ $# -ne 0 ]
do
    do_print $1
```

```
        shift
done
sun% mkdir -p /tmp/ja/LC_MESSAGES
sun% cp my_domain.mo /tmp/ja/LC_MESSAGES
sun% setenv TEXTDOMAINDIR /tmp
sun% chmod 755 my_script.sh
sun% locale
LANG=ja
LC_CTYPE="ja"
LC_NUMERIC="ja"
LC_TIME="ja"
LC_COLLATE="ja"
LC_MONETARY="ja"
LC_MESSAGES="ja"
LC_ALL=
sun% ./my_script.sh
使用方法: my_script 引数 ...
sun% env LANG=C ./my_script.sh
Usage: my_script ...
sun% ./my_script.sh 子 丑 寅
引数は 子
引数は 丑
引数は 寅
sun% env LANG=C ./my_script.sh Jan Feb Mar
arg is Jan
arg is Feb
arg is Mar
```

日本語専用ライブラリ (libjapanese.a)

JFP では、日本語特有の処理を行うために `libjapanese.a`¹ を提供しています。この章では次の項目について説明します。

注 - `libjapanese.a` およびそれに関連する次のヘッダファイルは、将来のリリースでは提供されません。新規アプリケーションには、前項までに説明した XPG などの標準関数を使用してください。すでに `libjapanese.a` を使用しているアプリケーションプログラムはソース互換性を保つための代替関数及びマクロを、ソースファイルにて提供します。詳細は、SUNWjlibj パッケージに含まれる (Entire インストールのみ含まれる) `/usr/share/src/libjapanese/README` を参照してください。

- 日本語専用ライブラリの使い方
- 各関数の簡単な説明と使用例

各関数に関する詳しい説明は、JFP 関係のマニュアルページを参照してください。

概説

JFP には、日本語特有の機能を実現する日本語専用ライブラリが含まれています。

一般に、このライブラリに含まれている関数は、プログラムのロケールがシステムで定義された `ja` またはそれに相当する日本語ロケールに設定されていることを前提

1. 「はじめに」でも述べましたが、ここで紹介するインタフェースは、`ja` (または `japanese`) ロケール上でのみ動作が保証されます。

としています。したがって、このライブラリの関数と SunOS 標準の国際化関数を同時に使ったプログラムが日本語データに関して一貫した動作をするためには、プログラムの始めに `setlocale(3C)` 関数が呼ばれ、これを通じてプログラムのロケールが適当な日本語ロケールに設定される必要があります。

また、日本語専用ライブラリには、JIS コード、EUC、PC 漢字 (MS 漢字、シフト JIS と呼ばれる) コード間などのコード変換を行う関数も定義されています。

次の関数を使用する場合には、コンパイル時に `-ljapanese)` オプションを指定して、日本語専用ライブラリをリンクしてください。

```
cjistosj(), cjistouj(), cujtojis(), cujtosj(), csjtojis(),  
csjtouj(), jis7tosj(), jis7touj(), sjtojis7(), ujtojis7(),  
jis8tosj(), jis8touj(), sjtojis8(), ujtojis8(), jistosj(),  
istouj(), ujtojis(), ujtosj(), sjtojis(), sjtouj, wstrtol(),  
wstrtod()
```

ワイド文字列処理関数

これらの関数のほとんどは、SunOS 5.6 の標準ライブラリ `libc` のワイド文字列処理関数に基づくマクロです。ここで説明する関数は、過去の日本語システムの日本語用ライブラリ仕様と互換性を保つために提供されています。これらの関数を使用するためには `widdec.h` ではなく、`/usr/include/wstring.h` を取り込んでください。ワイド文字列処理関数 (マクロ) を表 A-1 に示します。

表 A-1 ワイド文字列処理関数

関数	機能
<code>wstrcat()</code>	文字列を追加します
<code>wstrncat()</code>	指定文字数分だけ文字列を追加します
<code>wstricmp()</code>	文字列同士の比較を行います
<code>wstrncmp()</code>	指定文字数分だけ文字列同士の比較を行います

表 A-1 ワイド文字列処理関数 続く

関数	機能
<code>wstrcpy()</code>	文字列をコピーします
<code>wstrncpy()</code>	指定文字数分だけ文字列をコピーします
<code>wstrdup()</code>	<code>malloc</code> を使って文字列をコピーします
<code>wstrchr()</code>	指定文字が最初に現れた文字位置を返します
<code>wstrrchr()</code>	指定文字が最後に現れた文字位置を返します
<code>wstrspn()</code>	指定文字列で構成される最初の長さを返します
<code>wstrcspn()</code>	指定文字列以外で構成される最初の長さを返します
<code>wstrpbrk()</code>	指定文字列内の文字が最初に現れた文字位置を返します
<code>wstrlen()</code>	文字列の文字数を返します
<code>wstrtok()</code>	指定文字で分離される文字列を切り出します

また、次の表 A-2 で示す関数は、国際化機能に基づかない日本語固有の日本語処理を含んでおり、`libjapanese` で提供されます。

表 A-2 日本語固有のワイド文字列処理関数

関数	機能
<code>wstrtol()</code>	主および補助コードセット文字列を <code>long</code> 整数に変換します
<code>wstrtod()</code>	主および補助コードセット文字列を倍精度に変換します

コード例 A-1 に、このワイド文字列処理関数を使って、標準入力ファイル内の主および補助コードセットで書かれた数の合計を計算するプログラム例を紹介しま

す。このプログラムでは、`getws(3I)` 関数で行単位にファイルを読み込んだ後、`wstrpbrk(3X)` を操作させて整数を求めています。

コード例 A-1 ワイド文字列処理関数

```
sun% cat example1.c
#include <stdio.h>
#include <locale.h>
#include <wstring.h>

main()
{
    int total = 0;
    long token;
    wchar_t wbuf[256];
    wchar_t *wptr;

    static char *number = "01234567890 1 2 3 4 5 6 7 8 9";
    static wchar_t wnumber[21];

    setlocale(LC_ALL, "");
    if (mbstowcs(wnumber, number, 20) <= 0) {
        printf("数字列が認識できません。\\n");
        exit(1);
    }
    while (getws(wbuf) != NULL) {
        wptr = wbuf;
        while (wptr && *wptr) {
            wptr = wstrpbrk(wptr, wnumber);
            if (wptr) {
                token = wstrtoul(wptr, &wptr, 10);
                printf("トークンの値 = [%d]\\n", token);
                total += token;
            }
        }
    }
    printf("合計は %d です。\\n", total);
}
sun% cat 入力ファイル 1
343 34534
3 4 5 3 4 5 3 4 5 3 4 5
3 9 8 5 7 3 9 8 5 8 3 4 5 8 9
sun% cc -o example1 example1.c -lw -ljapanese
sun% ./example1 < 入力ファイル 1
トークンの値 = [343]
トークンの値 = [34534]
トークンの値 = [345345]
トークンの値 = [345345]
トークンの値 = [39857]
トークンの値 = [398]
トークンの値 = [5834589]
合計は 6600411 です。
sun%
```

ワイド文字変換用関数 (jconv)

日本語専用ライブラリでは、単一バイト変換関数 (toupper、tolower) と同様な変換関数を JIS X 0208、JIS X 0201、JIS X 0212 に相当する日本語文字専用に提供しています。さらに、ASCII 文字セットと日本語文字間での変換、区点コード変換など、日本語特有の変換を行う関数も提供しています。これらの関数を使用する場合は /usr/include/jctype.h を取り込んでください。

表 A-3 ワイド文字変換用関数

関数	機能
tojupper ()	JIS X 0208、JIS X 0212 で定義されるアルファベット小文字を表すワイド文字を対応する大文字を表すワイド文字に変換します
tojlower ()	JIS X 0208、JIS X 0212 で定義されるアルファベット大文字を表すワイド文字を対応する小文字を表すワイド文字に変換します
tojhira ()	JIS X 0208 で定義されるカタカナ文字、カタカナ繰返し記号を表すワイド文字を対応するひらがな文字、ひらがな繰返し記号を表すワイド文字に変換します
tojkata ()	JIS X 0208 で定義されるひらがな文字、ひらがな繰返し記号を表すワイド文字を対応するカタカナ文字、カタカナ繰返し記号を表すワイド文字に変換します
atojis ()	ASCII 文字、または JIS X 0201 カタカナ用図形文字で定義される文字を表すワイド文字を対応する JIS X 0208 で定義される文字を表すワイド文字に変換します
jistoa ()	JIS X 0208 で定義される文字を表すワイド文字を対応する ASCII 文字、または JIS X 0201 カタカナ用図形文字で定義される文字を表すワイド文字に変換します
toujis ()	ワイド文字表現に使用しないすべてのビットをオフにして、wchar_t 型に変換します
kutentojis ()	区点番号からワイド文字に変換します

コード例 A-2 に、これらのワイド文字変換用関数を使用して、JIS X 0208 のカタカナ文字をひらがな文字に変換するプログラム例を示します。このプログラム例で

は、`getws(3I)` 関数で行単位にファイルを読み込んだ後に、`tojhira(3X)` 関数を用いてカタカナ文字をひらがな文字に変換しています。

コード例 A-2 ワイド文字変換用関数

```
sun% cat example2.ccat example2.c
#include <stdio.h>
#include <locale.h>
#include <jctype.h>

main()
{
    wchar_t          wbuf[1024];
    wchar_t          *wptr, c;

    setlocale(LC_ALL, '');
    while (getws(wbuf) != NULL) {
        wptr = wbuf;
        while (*wptr) {
            c = tojhira(*wptr++);
            putwc(c, stdout);
        }
        putchar('\n');
    }
}
sun% cat input_file2cat input_file2
コレハ、カタカナヲひらがなニ変換スル
サンプルプログラムデス。
「ヴカケ」ハ、変換サレマセン。
クリカエシキゴウ - ヴ、
sun% cc -o example2 example2.c -lwcc -o example2 example2.c -lw
sun% ./example2 < input_file2
これは、かたかなをひらがなに変換する
さんぶるぷろぐらむです。
「ヴカケ」は、変換されません。
くりかえしきごう - ヴ、
sun%
```

ワイド文字分類関数 (jctype)

日本語専用ライブラリでは、文字を分類する機能を提供しています。これらの関数を使用する場合は、`/usr/include/jctype.h` を取り込んでください。

表 A-4 ワイド文字分類関数

関数	機能
<code>isjis()</code>	JIS X 0208 で定義される文字を表すワイド文字のとき、真を返します
<code>isjalpha()</code>	JIS X 0208 で定義される英語のアルファベット文字を表すワイド文字のとき、真を返します
<code>isjupper()</code>	JIS X 0208、JIS X 0212 で定義されるアルファベットの大文字を表すワイド文字のとき、真を返します
<code>isjlower()</code>	JIS X 0208、JIS X 0212 で定義されるアルファベットの小文字を表すワイド文字のとき、真を返します
<code>isjdigit()</code>	JIS X 0208 で定義される数字を表すワイド文字のとき、真を返します
<code>isjspace()</code>	JIS X 0208 で定義される空白文字を表すワイド文字のとき、真を返します
<code>isjpunct()</code>	JIS X 0208、JIS X 0212 で定義される記述記号を表すワイド文字のとき、真を返します
<code>isjparen()</code>	JIS X 0208 で定義される括弧記号を表すワイド文字のとき、真を返します
<code>isjline()</code>	JIS X 0208 で定義される罫線けい線素片を表すワイド文字のとき、真を返します
<code>isjunit()</code>	JIS X 0208 で定義される単位記号を表すワイド文字のとき、真を返します
<code>isjsoci()</code>	JIS X 0208 で定義される学術記号を表すワイド文字のとき、真を返します
<code>isjgen()</code>	JIS X 0208、JIS X 0212 で定義される一般記号を表すワイド文字のとき、真を返します
<code>isjkanji()</code>	JIS X 0208、JIS X 0212 で定義される漢字を表すワイド文字のとき、真を返します
<code>isjspecial()</code>	JIS X 0208 で定義される特殊文字を表すワイド文字のとき、真を返します
<code>isjgreek()</code>	JIS X 0208 で定義されるギリシャ文字を表すワイド文字のとき、真を返します

表 A-4 ワイド文字分類関数 続く

関数	機能
<code>isjruussian()</code>	JIS X 0208 で定義されるロシア文字を表すワイド文字のとき、真を返します
<code>isjkata()</code>	JIS X 0208 で定義されるカタカナ、濁点、半濁点、長音記号、カタカナ繰り返し記号を表すワイド文字のとき、真を返します
<code>isjhira()</code>	JIS X 0208 で定義されるひらがな、濁点、半濁点、長音記号、ひらがな繰り返し記号を表すワイド文字のとき、真を返します
<code>isj1bytekana()</code>	JIS X 0201 で定義されるカタカナ用図形文字に属する文字を表すワイド文字のとき、真を返します
<code>isjhankana()</code>	<code>isj1bytekana()</code> と同様

次に、これらのワイド文字分類関数を使用して、JIS X 0208 の英語のアルファベット文字と数字を、ASCII の英語のアルファベットと数字に変換するプログラムを紹介いたします。このプログラム例では、`getws(3I)` 関数で行単位にファイルを読み込んだ後に、`isjalpha(3X)` と `isjdigit(3X)` でワイド文字を分類し、`jistoa(3X)` 関数を用いて ASCII に変換しています。

コード例 A-3 ワイド文字分類関数

```
sun% cat example3.c
#include <stdio.h>
#include <locale.h>
#include <jctype.h>

main()
{
    wchar_t      wbuf[1024];
    wchar_t      *wptr;

    setlocale(LC_ALL, "");
    while (getws(wbuf) != NULL) {
        wptr = wbuf;
        while (*wptr) {
            if (isjdigit(*wptr) || isjalpha(*wptr)){
                printf("%'wc'", jistoa(*wptr));
            }else {
                printf("%'wc'", *wptr);
            }
            *wptr++;
        }
    }
}
```

```

        }
        putchar('\n');
    }
}
sun% cat input_file3
Sun Microsystems
0123456789
sun% cc -o example3 example3.c -lw -ljapanese
sun% ./example3 < input_file3
Sun Microsystems
0123456789
sun%

```

x86: コード変換用関数 (jisconv、ibmjcode)

日本語専用ライブラリでは、JIS X 0208 文字集合のコード変換機能を提供しています。これらの関数は、JIS X 0208 (jis)、日本語 EUC コード (uj)、PC 漢字コード (sj) 間でコードの変換を行うものです。これらの関数を使用する場合は、`/usr/include/jcode.h` を取り込んでください。cjistosj (3X) や cujtosj (3X) などの 1 文字コード変換用関数は、JIS X 0208 の漢字セット、および JIS X 0201 のアルファベット文字またはかな文字のセットで定義され、JIS X 0208-1983 文字セットの指示は「ESC\$B」、JIS X 0201-1976 文字セットの指示は「ESC (J)」のシーケンスによって、それぞれ行われます。

表 A-5 コード変換用関数

関数	機能
cjistosj()	漢字の JIS 文字 1 文字を PC 漢字文字に変換します
cjistouj()	漢字の JIS 文字 1 文字を日本語 EUC コード文字に変換します
cujtojis()	漢字の日本語 EUC コード文字 1 文字を JIS 文字に変換します
cujtosj()	漢字の日本語 EUC コード文字 1 文字を PC 漢字文字に変換します
csjtojis()	PC 漢字文字 1 文字を JIS 文字に変換します
csjtouj()	PC 漢字文字 1 文字を日本語 EUC コード文字に変換します

表 A-5 コード変換用関数 続く

関数	機能
<code>jis7tosj()</code>	7 ビット JIS 文字の文字列を PC 漢字文字の文字列に変換します
<code>jis7touj()</code>	7 ビット JIS 文字の文字列を日本語 EUC コード文字の文字列に変換します
<code>sjtojis7()</code>	PC 漢字文字の文字列を 7 ビット JIS 文字の文字列に変換します
<code>ujtojis7()</code>	日本語 EUC コード文字の文字列を 7 ビット JIS 文字の文字列に変換します
<code>jis8tosj()</code>	8 ビット JIS 文字の文字列を PC 漢字文字の文字列に変換します
<code>jis8touj()</code>	8 ビット JIS 文字の文字列を日本語 EUC コード文字の文字列に変換します
<code>sjtojis8()</code>	PC 漢字文字の文字列を 8 ビット JIS 文字の文字列に変換します
<code>ujtojis8()</code>	日本語 EUC コード文字の文字列を 8 ビット JIS 文字の文字列に変換します
<code>jistosj()</code>	<code>jis8tosj()</code> と同じです
<code>jistouj()</code>	<code>jis8touj()</code> と同じです
<code>ujtojis()</code>	<code>ujtojis8()</code> と同じです
<code>ujtosj()</code>	日本語 EUC コード文字の文字列を PC 漢字文字の文字列に変換します
<code>sjtojis()</code>	<code>sjtojis8()</code> と同じです
<code>sjtouj()</code>	PC 漢字文字の文字列を日本語 EUC コード文字の文字列に変換します

コード例 A-4 に、このコード変換用関数を用いて、EUC から JIS コードへコード変換するプログラム例を示します。

コード例 A-4 コード変換用関数

```
sun% cat example4.c
#include <stdio.h>
#include <jcode.h>

main()
{
    char inbuf[2048];
    char outbuf[2048];

    while(gets(inbuf) != NULL) {
        ujtojis(outbuf, inbuf);
        puts(outbuf);
    }
}
sun% cc -o example4 example4.c -ljapanese
sun% cat 入力ファイル4
これは、コード変換のテスト用に作成したファイルです。
いかがでしょう。
sun% ./example4 < 入力ファイル4 | jistoec
これは、コード変換のテスト用に作成したファイルです。
いかがでしょう。
sun%
```

また、IBM 漢字コードと日本語 EUC コード間のコード変換を行うため、一連の関数が提供されています (ibmjcode(3X) 参照)。これらの関数を使用する場合には、`/usr/include/ibmjcode.h` を取り込んでください。

Solaris の日本語 TrueType フォント

この章では、Solaris で提供する TrueType フォントと、そのインストール方法などについて説明します。

TrueType フォントのサポート

Solaris では、一般に入手できる TrueType フォントを X, DPS から使用できます。サポートしているフォントの Platform ID および Encoding ID はそれぞれ以下のとおりです。

表 B-1 Solaris でサポートしている TrueType フォント

Platform ID	Encoding ID
3 (Microsoft)	1 (Unicode)
	2 (ShiftJIS)

ただし、DPS からは Encoding ID が 2 (ShiftJIS) のフォントは使用できません。

Solaris で提供する日本語 TrueType フォント

Solaris では、株式会社リコー (以下、リコーとします) が開発した HG ゴシック体 B、HG 明朝体 L、平成明朝体 W3H を提供しています。また、X, DPS 上で小さいサイズのグリフ表示時のパフォーマンス向上のため、それぞれのフォントに対応した各種サイズのビットマップフォントも同時に提供されています。

- HG ゴシック体 B
 - XLFD 名: -ricoh-hg gothic b-medium-r-normal-*
 - DPS でのフォント名: HG-GothicB
- HG 明朝体 L
 - XLFD 名: -ricoh-hg mincho l-medium-r-normal-*
 - DPS でのフォント名: HG-MinchoL
- 平成明朝体 W3H
 - XLFD 名: -heiseimin-w3-r-normal-*
 - DPS でのフォント名: HiseiMin-W3H

HG ゴシック体 B、HG 明朝体 L は JIS X 0201, JIS X 0208 文字セット用のフォントです。また平成明朝体 W3H は JIS X 0212 補助漢字用のフォントです。平成明朝体 W3H フォントは、Sun Gothic、Sun Mincho、HG ゴシック体 B、HG 明朝体 L と組み合わせて使われます。

TrueType フォントおよび対応するビットマップフォントは以下のディレクトリにインストールされています。

- TrueType フォント
`/usr/openwin/lib/locale/ja/X11/fonts/TT`
- TrueType ビットマップフォント
`/usr/openwin/lib/locale/ja/X11/fonts/TTbitmaps`

Solaris 2.5.1 以前のシステムとのフォントの互換性

リコーの TrueType フォントおよび TrueType ビットマップフォントの提供に伴い、Solaris 2.5.1 以前のシステムで提供されていた、株式会社モリサワの F3 アウトラインフォント、F3 ビットマップフォント (以下、モリサワフォントとします) および東京大学和田研究室漢字分科会の補助漢字用 Type1 フォント、ビットマップフォント (以下、和田研フォントとします) は提供されなくなりました。なお、これらのフォントを用いたアプリケーションの互換性を保つため、リコーのフォントを使用するよう別名を定義してあります。

X から使われるフォント

- モリサワフォント
 - -morisawa-gothic medium bbb-bold-r-normal-sans-*
 - -morisawa-gothic medium bbb-medium-r-normal-sans-*
 - -ricoh-hg gothic b-medium-r-normal-* を使用します。
 - -morisawa-rymin light kl-light-r-normal-*
 - -ricoh-hg mincho l-medium-r-normal-* を使用します。
- 和田研フォント
 - -wadalab-gothic-medium-r-normal-*
 - -wadalab-mincho-medium-r-normal-*
 - -wadalab-gothic-bold-r-normal-*
 - -wadalab-mincho-bold-r-normal-*
 - -ricoh-heiseimin-w3- を使用しています。

DPS から使われるフォント

- モリサワフォント

- GothicBBB-Medium
- GothicBBB-Medium-Bold
HG-GothicB を使用します。
- Ryumin-Light
HG-MinchoL を使用します。

Soralis 2.5.1 からの変更点

Soralis では、米国 Adobe System, Incorporated (以降 Adobe とします) 純正の CMap 使ってます。このため、Solaris 2.5.1 以前のシステムで提供されていた以下のフォント、区フォント、エンコーディングは提供されません。

表 B-2 提供されないフォント

Bpld-H	78-SuppA-H	78-SuppA-V	78-SuppB-HV
83pv-SuppA-H	83pv-SuppB-H	Ext-SuppA-H	Ext-SuppA-V
Ext-SuppB-HV	SuppA-H	SuppA-V	SuppB-HV
UserGaiji	SuppK	jisx0201	

表 B-3 提供されない区フォント

78.r30	78.r31	78.r32	78.r33
78.r34	78.r35	78.r36	78.r37
78.r38	78.r39	78.r3A	78.r3B
78.r3C	78.r3D	78.r3E	78.r3F
78.r40	78.r41	78.r42	78.r43
78.r44	78.r45	78.r46	78.r47
78.r48	78.r49	78.r4A	78.r4B
78.r4C	78.r4D	78.r4E	78.r4F
78.r50	78.r51	78.r52	78.r53
78.r54	78.r55	78.r56	78.r57
78.r58	78.r59	78.r5A	78.r5B
78.r5C	78.r5D	78.r5E	78.r5F
78.r60	78.r61	78.r62	78.r63
78.r64	78.r65	78.r66	78.r67
78.r68	78.r69	78.r6A	78.r6B
78.r6C	78.r6D	78.r6E	78.r6F
78.r70	78.r71	78.r72	78.r73
78.sr88	78.sr89	78.sr8A	78.sr8B
78.sr8C	78.sr8D	78.sr8E	78.sr8F
78.sr90	78.sr91	78.sr92	78.sr93
78.sr94	78.sr95	78.sr96	78.sr97
78.sr98	78.sr99	78.sr9A	78.sr9B
78.sr9C	78.sr9D	78.sr9E	78.sr9F
78.srE0	78.srE1	78.srE2	78.srE3
78.srE4	78.srE5	78.srE6	78.srE7
78.srE8	78.srE9	78.srEA	Ext.r21
Ext.r21v	Ext.r22	Ext.r22v	Ext.r22w
Ext.r23	Ext.r29	Ext.r2A	Ext.r2B
Ext.r2C	Ext.r2D	Ext.r2Dv	Ext.r2E
Ext.r2F	Ext.r2Fv	Ext.r30	Ext.r31
Ext.r3C	Ext.r44	Ext.r49	Ext.r4B
Ext.r53	Ext.r56	Ext.r5F	Ext.r69
Ext.r6F	Ext.r70	Exr.sr81	Ext.sr81v
Ext.sr82	Ext.sr82V	Exr.sr84	Ext.sr85
Ext.sr86	Ext.sr87	Ext.sr87v	Ext.sr88
Ext.sr89	Ext.sr8E	Ext.sr92	Ext.sr96
Ext.sr9A	Ext.sr9B	Ext.srE0	Ext.srE5
Ext.srE8	r21	r21v	r22
r23	r24	r24v	r25
r25v	r26	r27	r28
r30	r31	r32	r33
r34	r35	r36	r37
r38	r39	r3A	r3B
r3C	r3D	r3E	r3F
r40	r41	r42	r43
r44	r45	r46	r47
r48	r49	r4A	r4B
r4C	r4D	r4E	r4F
r50	r51	r52	r53
r54	r55	r56	r57
r58	r59	r5A	r5B
r5C	r5D	r5E	r5F

r60	r61	r62	r63
r64	r65	r66	r67
r68	r69	r6A	r6B
r6C	r6D	r6E	r6F
r70	r71	r72	r73
r74	sr81	sr81v	sr82
sr82v	sr83	sr83v	sr84
sr88	sr89	sr8A	sr8B
sr8C	sr8D	sr8E	sr8F
sr90	sr91	sr92	sr93
sr94	sr95	sr96	sr97
sr98	sr99	sr9A	sr9B
sr9C	sr9D	sr9E	sr9F
srE0	srE1	srE2	srE3
srE4	srE5	srE6	srE7
srE8	srE9	srEA	

表 B-4 提供されないエンコーディング

ExtJIS12-88-CFEncoding	ExtJIS12-97-CFEncoding
ExtShiftJIS-A-CFEncoding	ExtShiftJIS12-88-CFEncoding
JIS12-88-CFEncoding	JIS12-97-CFEncoding
ShiftJIS-A-CFEncoding	ShiftJIS-B-CFEncoding
ShiftJIS12-88-CFEncoding	

また、フォントを Type0 フォントと仮定して FDepVector を利用し、フォントを再定義するような PostScript (TM) プログラムは作動されません。Solaris 2.5.1 以前にフォントマネージャーなどで作成されたユーザー定義文字フォント、Solaris 2.5.1 以前の mp (1) やカレンダーマネージャで出力された PostScript は DPS で表示することはできません。

TrueType フォントのインストール方法

市販の日本語 TrueType フォントをインストールする場合には、以下の方法で行ってください。

1. TrueType フォントを適切なディレクトリにコピーします。なお、Solaris では TrueType Collection (.ttc) フォントは直接使用することができません。TrueType Collection フォントは複数の TrueType フォントに分割して使用するようになっています。

```
% mkdir $HOME/ttfontdir
% cp sample.ttf $HOME/ttfontdir
```

なお、フォントを置く場合は任意のディレクトリでかまいません。

2. `fonts.scale` ファイルを作成します。 `fonts.scale` のフォーマットは `fonts.dir` ファイルと同じです。詳しくは `mkfontdir(1)` を参照してください。フォントの **XLFD (X Logical Font Description)** のフォーマットですが、**PIXEL_SIZE, POINT_SIZE, RESOLUTION_X RESOLUTION_Y** および **AVERAGE_WIDTH** フィールドでは全て **0** にします。また **SPACING field** は、固定幅フォントの場合は **'m'** に、プロポーショナルフォントの場合には **'p'** にしてください。 **CHARSET_REGISTRY** および **CHARSET_ENCODING** フィールドは、一般の日本語 **TrueType** フォントの場合 **'jisx0208.1983-0'**, **'jisx0201.1976-0'**, **'jisx0212.1990-0'** の 3 通りを指定できます。ただし、一般の日本語 **TrueType** フォントでは、**JIS X 0212** 補助漢字の全てのグリフをサポートしておらず、**Microsoft** 標準キャラクタセットの **IBM** 拡張文字にあるいくつかのグリフのみ使用できます。

■ 例 1 固定幅フォントの場合

```
sample.ttf -sample-misc-medoum-r-normal--0-0-0-0-m-0-jisx0208.1983-0
sample.ttf -sample-misc-medoum-r-normal--0-0-0-0-m-0-jisx0201.1976-0
sample.ttf -sample-misc-medoum-r-normal--0-0-0-0-m-0-jisx0212.1990-0
```

■ 例 2 プロポーショナルフォントの場合

```
sample.ttf -sample-misc-medoum-r-normal--0-0-0-0-p-0-jisx0208.1983-0
sample.ttf -sample-misc-medoum-r-normal--0-0-0-0-p-0-jisx0201.1976-0
sample.ttf -sample-misc-medoum-r-normal--0-0-0-0-p-0-jisx0212.1990-0
```

3. `mkfontdir` コマンドで `fonts.dir` ファイルを作成します。

```
% /usr/openwin/bin/mkfontdir
```

4. フォントパスを追加します。

■ 例 1 一時的フォントパスを追加する場合

```
% /usr/openwin/bin/xset/ fp+ $HOME/ttfontdir
```

■ 例 2 各ユーザーの設定ファイルを書き換える場合

`$HOME/.OWfontpath` にフォントの存在するディレクトリパスを追加し、ウィンドウシステムを再起動します。

注 - `OWfontpath` の仕様は将来変更される可能性があります。この用途以外で変更を行った場合に動作は保証されません。

■ 例 3 システムの設定ファイルを書き換える場合

`/usr/openwin/lib/locale/<locale>/OWfontpath` にフォントの存在するディレクトリを追加し、ウィンドウシステムを再起動します。

注 - `OWfontpath` の仕様は将来変更される可能性があります。この用途以外で変更を行った場合の動作は保証されません。

索引

C

CSI (Code Set Independence), 2

E

EUC (Extended UNIX Code), 2

I

ibmjcode(), 51
isj1bytekana(), 50, 52
isjalpha(), 47, 49, 51
isjdigit(), 47, 49, 51
isjgen(), 49, 52
isjgreek(), 49, 52
isjhira(), 50, 52
isjis(), 47, 49, 51
isjkanji(), 49, 52
isjkata(), 50, 52
isjline(), 49, 52
isjlower(), 47, 49, 51
isjparen(), 47, 49, 52
isjpunct(), 47, 49, 52
isjrussian(), 50, 52
isjsci(), 49, 52
isjspace(), 47, 49, 51
isjspecial(), 49, 52
isjunit(), 49, 52
isjupper(), 47, 49, 51

J

ja_JP.UTF-8, 3
jconv(), 47
jctype(), 48
JFP(Japanese Feature Package), 1, 5
jisconv(), 51

L

libjapanese.a, 43
libw, 44

M

MS 漢字, 44

N

NLSPATH 環境変数, 38

P

PCK (PC 漢字コード), 3
PC 漢字, 44

W

wstrcat(), 44
wstrchr(), 45

wstrcmp(), 44
wstrncpy(), 45
wstrcspn(), 45
wstrdup(), 45
wstrlen(), 45
wstrncat(), 44
wstrncmp(), 44
wstrncpy(), 45
wstrpbrk(), 45
wstrrchr(), 45
wstrspn(), 45
wstrtod(), 45
wstrtok(), 45
wstrtol(), 45

X

X/Open 方式, 36
X/Open 方式, 36
XPG, 2, 5, 17, 27

か

環境変数, 1

け

言語対応化, 1

こ

国際化プログラミング, 1
コードセット, 2
コード変換用関数, 51
国際化, 1, 31
国際化 API, 6, 9

し

シェルスクリプト, 39

シフト JIS, 44

て

テキストドメイン, 40

は

汎用コード変換 API, 30

ふ

複数バイト表現, 6, 27
複数バイト文字列, 6

ほ

ポータブル・オブジェクト, 40

め

メッセージ・オブジェクト, 41
メッセージカタログファイル, 36
メッセージカタログファイル記述子, 37
メッセージ処理, 35

も

文字集合, 2

わ

ワイド文字表現, 9, 27, 39
ワイド文字列, 10
ワイド文字分類関数, 48
ワイド文字変換用関数, 47
ワイド文字列処理関数, 44