JFP 開発ガイド

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

Part Number 806–2802–10
2000 年 3 月

Adobe PostScript

目次

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方のために、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、『国際化対応言語環境の利用ガイド』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ついて解説します。

- 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 (このマニュアルでは C 言語とシェルスクリプトを使って説明します)
- 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

内容の紹介

このマニュアルは次の章から構成されています。

第 1 章

マニュアルを読み進むにあたって必要な概念、または知っていると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 で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と、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と `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ています。

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。

前もって読む必要があるマニュアル

- 『*Solaris 8* ご使用にあたって』
- 『*JFP* ユーザーズガイド』

関連マニュアル・書籍

- 『*SunOS* リファレンスマニュアル』
- 『*man pages section 3*』
- 『システムインタフェース』
- 『*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*』

Sun のマニュアルの注文方法

専門書を扱うインターネットの書店 Fatbrain.com から、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) 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、`http://www1.fatbrain.com/documentation/sun` の Sun Documentation Center をご覧ください。

Sun のオンラインマニュアル

http://docs.sun.com では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索をおこなうこともできます。

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、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.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。	『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。

表 P–1 表記上の規則 続く

字体または記号	意味	例
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

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。

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 y|n [filename]
```

- Bourne シェルおよび Korn シェルのプロンプト

```
system$ command y|n [filename]
```

- スーパーユーザーのプロンプト

```
system# command y|n [filename]
```

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

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

一般規則

- このマニュアルでは、「IA」という用語は、Intel 32 ビットのプロセッサアーキテクチャを意味します。これには、Pentium、Pentium Pro、Pentium II、Pentium II Xeon、Celeron、Pentium III、Pentium III Xeon の各プロセッサ、および AMD、Cyrix が提供する互換マイクロプロセッサチップが含まれます。

第 1 章

概要

この章では、マニュアルを読み進むにあたって必要な概念、または知っていると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・実装する手法のことです。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」と呼ばれる作業を経て、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て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のことです。

『*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 1 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。このマニュアルも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、時刻などの情報を各地域・各文化に適合させて動作するようになります。環境変数を使用することで、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)` と `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。この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、実際の動作が 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)` のマニュアルページに記載されている手順に従ってコンパイルとリンクを行う必要があります。各リリースに対応するXPG のバージョンについても、`standards(5)` のマニュアルページ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が重要です。しかし、日本をはじめとするアジア系の地域では、使用する文字数が多いので、すべての文字の文字コードを格納して表現するためには複数バイト必要です。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 種には、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 と ja_JP.eucJP ロケールにおける日本語 EUC の定義と文字集合について説明しています。

CSI

国際化の基礎となる文字コード体系に EUC を採用した場合、EUC 以外の文字コード体系を使って言語対応化することは、原則としてできません。この問題を解消す

るため、Solaris 2.5.1 以降のバージョンでは、システムから EUC に依存する部分を取り除き、 X/Open 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 (Code Set Independence。以降「CSI」と記述) を導入しました。CSI の下では、EUC も文字コード体系の 1 つとして扱われます。

CSI の導入に伴って、Solaris 2.5.1 からは「PCK」、Solaris 7 からは UTF-8 がサポートされるようになりました。

『*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 と、ja_JP.UTF-8 ロケールにおける UTF-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が提供されています。1 つは、日本語固有の言語情報と文化情報です。これらは、国際化されたプログラムを支援する言語対応化の 1 つであり、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可能です。この枠組みに従って処理を行う限り、アプリケーションの高い移植性を保ちながら日本語化を行うことができます。

Solaris で提供するもう 1 つの日本語ロケール情報は、国際化の枠組みの中で日本語化を補う付加的な情報です。これらを利用すると、日本語に固有の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 以外の環境では、アプリケーションを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 章で紹介しています。

第 2 章

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」とは、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。この環境では、基になる Solaris が XPG に適合しているため、XPG で規定されているすべてのインタフェースを利用できます。XPG には、国際化プログラミングに必要な API が豊富に用意されており、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 を介して文字列を処理し、アプリケーションを開発することができます。このとき、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する必要はありません。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 の仕様の詳細については、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 に、複数バイト表現の文字列操作に使用する主な国際化 API を紹介します。API は、これ以外にも用意されています。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
`strcat(s1,s2)`	`s2` を `s1` に追加する。追加後の `s1` が返る
`strncat(s1,s2,n)`	`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
`strcmp(s1,s2)`	`s1` と `s2` の大小関係を調べる (順序情報に基づかない)
`strncmp(s1,s2,n)`	`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)
`strcoll(s1,s2)`	順序情報に基づき `s1` と `s2` の大小関係を調べる
`strcpy(s1,s2)`	`s2` を `s1` にコピーする。コピー後の `s1` が返る
`strncpy(s1,s2,n)`	`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
`strlen(s)`	`s` の長さをバイト数で返す
`strxfrm(s1,s2,n)`	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()` と `strrchr()` がありますが、これらの API は複数バイト文字列に対して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()` 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

ここでは、複数バイト文字列操作 API を使用したプログラム例を紹介します。複数バイト文字列操作 API を使用する場合は、string.h ヘッダーファイルを取り込み、処理の最初の段階で setlocale() を呼び出して、動作ロケールを適切に設定する必要があります。

次の例では、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。順序情報に基づいて比較をする場合は 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() で比較する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われます。データベースを管理する場合など、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。

例 2–1 では、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まったく同じ結果が得られます。

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 artificialy 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
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

表 2–2 に、ワイド文字表現の文字列操作に使用する主な国際化 API を紹介します。API は、これ以外にも用意されています。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
wcscat(ws1,ws2)	ws2 を ws1 に追加する。追加後の ws1 が返る
wcsncat(ws1,ws2,n)	ws2 のうち最大 n ワイド文字を ws1 に追加する。追加後の ws1 が返る
wcschr(ws,wc)	ws の先頭から調べ、最初に出現する wc の位置が返る
wcsrchr(ws,wc)	ws の先頭から調べ、最後に出現する wc の位置が返る
wcscmp(ws1,ws2)	ws1 と ws2 の大小関係を調べる (順序情報に基づかない)
wcsncmp(ws1,ws2,n)	ws1 と ws2 の最大 n ワイド文字の大小関係を調べる (順序情報に基づかない
wcscoll(ws1,ws2)	順序情報に基づき ws1 と ws2 の大小関係を調べる
wcscpy(ws1,ws2)	ws2 を ws1 にコピーする。コピー後の ws1 が返る
wcsncpy(ws1,ws2,n)	ws2 の最大 n ワイド文字を ws1 にコピーする。コピー後の ws1 が返る
wcslen(ws)	ws の長さをワイド文字数で返す
wcsxfrm(ws1,ws2,n)	大小関係を調べるための文字列の変形
wcwidth(wc)	wc の表示に必要なカラム数
wcswidth(ws, n)	ws 中の最大 n ワイド文字の表示に必要なカラム数

プログラム例

ここでは、ワイド文字列操作 API を使用したプログラム例を 2 つ紹介します。ワイド文字列操作 API を使用する場合は、wchar.h ヘッダーファイルを取り込み、処理の最初の段階で setlocale() を呼び出して、動作ロケールを適切に設定する必要があります。

例 2–2 では、ワイド文字列を比較しています。例 2–1 で、複数バイト表現の例として strxfrm() と strcmp() を使って 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のプログラミング例を紹介します。

例 2–2 では、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まったく同じ結果が得られます。

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 artificialy 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_wcscoll((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_wcscoll(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_wcscoll(): Error in transforming 1st string'');
  exit(-1);
 }
 wcsxfrm(transform_2, wcp2, (xfrm_len2 + 1));
 if (errno != 0) {
  perror(``my_wcscoll(): 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
```

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。表示の際は、1 行あたり 80 カラムを最大とし、入力ファイルの行末が均等に削除されます。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。

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 += wcwidth(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 = wcwidth(*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 = wcwidth(*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
01234567890123456789012345678901234567890123456789012345678901234567890123456789
０１２３４５６７８９０１２３４５６７８９０１２３４５６７８９０１２３４５６７８９
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
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。
０１２３４５６７８９０１２３４５６７８ | 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 |
```

第 3 章

日本語ロケールと文字分類

この章では、XPG で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と、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 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) を支援します。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 を介して簡単に大文字・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と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 の 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 章を参照してください。

日本語文字対応

文字対応の操作に使用する 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。API は、これ以外にも用意されています。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
`towupper (wc)`	ワイド文字 wc に対応する大文字ワイド文字表現を返す。なければ wc をそのまま返す
`towlower (wc)`	ワイド文字 wc に対応する小文字ワイド文字表現を返す。なければ wc をそのまま返す
`towctrans (wc, wctrans("タグ名"))`	タグ名に基づいてワイド文字 wc に対応したワイド文字表現を返す。なければ wc をそのまま返す
`wctrans ("タグ名")`	`towctrans ()` で使うタグ名に対応する値を返す。以下のタグ名を使用できる "tolower" "toupper"

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
`wctrans("タグ名")`	`towctrans()` で使うタグ名に対応する値を返す。Solaris の日本語ロケールでは、以下のタグ名を使用できる "tojhira" "tojkata" "tojisx0208" "tojisx0201"

プログラム例

ここでは、日本語文字対応の API を使用したプログラム例を 2 つ紹介します。日本語文字対応 API を使用する場合は、wchar.h、wctype.h ヘッダーファイルを取り込み、処理の最初の段階で setlocale() を呼び出して、動作ロケールを適切に設定する必要があります。

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、通信に先立って、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。

例 3–1 日本語文字対応 API

```
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。

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。

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 <wtype.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                                              12
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 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。API は、これ以外にも用意されています。詳しくはマニュアルページ (`iswalpha(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
`iswctype(wc, type)`	wc が type クラスに属するかどうか調べる

`wctype("タイプ名")`	`iswctype()` 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
	
	`"alnum" "alpha" "cntrl" "digit" "graph" "lower"`
	
	`"print" "punct" "space" "upper"`
	
	`"xdigit" "blank"`

表 3–3　文字分類 API その 1　続く

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")	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 "jkanji" "jkata" "jhira" "jdigit" "jparen" "jline" "jisx0201r" "jisx0208" "jisx0212" "udc" "vdc" "jalpha" "jspecial" "jgreek" "jrussian" "junit" "jsci""jgen" "jpunct"

プログラム例

ここでは、日本語文字分類 API を使用したプログラム例を紹介します。この場合も前述の文字対応 API の場合と同様に、wchar.h ヘッダーファイルを取り込み、処理の最初の段階で setlocale() を呼び出して、動作ロケールを適切に設定する必要があります。

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 数字文字を ASCII に変換して出力します。

例 3–3　文字分類 API

```
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ａｌｐｈａｂｅｔ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ａｌｐｈａｂｅｔヤ
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。

第 4 章

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用インタフェースと、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 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、変換可能なすべてのエンコーディングに関する情報がアプリケーションに含まれて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 はこの用途の API を用意しており、汎用の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 に、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。このほか、printf(3C)、scanf(3C) 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
mbtowc(pwc,s,n)	s の先頭から最大 n 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 pwc へ格納
mbstowcs(pwcs,s,n)	s 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 n ワイド文字変換したら終了
wctomb(s,wc)	ワイド文字 wc を複数バイト表現に変換し s へ格納
wcstombs(s,pwcs,n)	pwcs からワイド文字列を複数バイト表現に変換しながら s に格納。変換した複数バイトの合計が最大 n バイトになれば終了
mblen(s,n)	s の先頭から最大 n 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
fgetwc(stream)	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
ungetwc(wc, stream)	ワイド文字 wc を stream へプッシュバックする
fgetws(ws,n,stream)	入力ストリーム stream から複数バイト文字列を読み込み、最大 n-1 ワイド文字分を ws に格納する
fputwc(wc,stream)	出力ストリーム stream へワイド文字 wc を出力
fputws(ws,stream)	出力ストリーム stream へワイド文字列 ws を出力

プログラム例

例 4–1 は、あるファイルに対して複数バイト・ワイド文字の相互変換 API を適用するプログラム例です。これらの API を使用する場合は、適切なヘッダーファイルを取り込む必要があります。たとえば `mbtowc()`、`mbstowcs()`、`wctomb()`、`wcstombs()`、`mblen()` を使用する場合は `stdlib.h` を取り込み、`ungetwc()`、`fgetws()`、`fputwc()`、`fputws()` を使用する場合は `wchar.h` を取り込みます。さらに、処理の最初の段階で `setlocale()` を呼び出し、動作ロケールを適切に設定する必要があります。

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 mbbuf[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(mbbuf, BUFSIZ, stdin) != NULL) {
                retval = mbstowcs(wcbuf, mbbuf, 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"))) {
 kana_char++;
```

```
} else if (iswctype(wcbuf[i], wctype("jkanji"))) {
 ideo_char++;
                        } else if (iswctype(wcbuf[i], wctype("alpha"))) {
                                alpha_char++;
                        } else {
                                other_char++;
                        }
                }
        }
        fprintf(stdout, "The input consist of\n");
        fprintf(stdout, "%d Alphabetical chars,\n", alpha_char);
        fprintf(stdout, "%d JIS X 0208/0212 Kanji chars,\n", ideo_char);
        fprintf(stdout, "%d JIS X 0201 Kana chars and\n", kana_char);
        fprintf(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ａｌｐｈａｂｅｔ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 0208/0212 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と、『*JFP* ユーザーズガイド』の第 6 章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
iconv_open()	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
iconv()	得られた情報をもとに実際の変換を行う
iconv_open()	変換に必要だった情報を解放する

プログラム例

例 4–2 では、iconv() インタフェースを使用して 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 ヘッダー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、変換前と変換後の文字集合とエンコーディングに関する情報を変換記述子を通して取得します。したがって、次のプログラム例でも setlocale() は呼び出されていません。

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。

第 5 章

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、表示させる方法を説明します。

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た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。

代わりに、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用意するだけで、アプリケーション中のテキストを容易に翻訳できます。

Solaris では、メッセージ処理の API が 2 種類提供されています。1 つは米国 Sun Microsystems, Inc. 独自のインタフェースである `gettext()` を使用する方式で、もう 1 つは X/Open で規定された `catgets()` を使用する方式です。これらの間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になる場合には、X/Open で規定されている方式でメッセージ処理を行う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概説し、その手順に従って X/Open 方式でのメッセージ処理 API について説明します。

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。

   これは、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. 46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むことで行います。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. 47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。

ここで、カタログファイルを利用するには、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 環境変数と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 ヘッダーファイルを取り込み、処理の最初の段階で setlocale を呼び出し、動作ロケールを適切に設定する必要があります。

```
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。

第 6 章

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 は、XPG 準拠の国際化されたコマンド・ユーティリティを多数提供しています。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を容易に作成できます。また、Solaris のメッセージ用コマンドを利用して、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の動作ロケールを正しく設定し、各コマンド・ユーティリティが処理を行えるような環境を用意する必要があります。この処理は、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、表示させるには、次の手順に従ってください。なお、第 5 章で紹介した catgets() を使用する場合とは、翻訳メッセージを格納するファイルの構造が異なっていることに注意してください。

1. テキストドメインと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 などで出力するメッセージは、 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と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使用されるほか、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、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には、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*" という行によって区切られます。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くなります。

```
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。これらのファイル名には、msgfmt コマンドによって接尾辞 .mo が付けられます。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、52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 TEXTDOMAIN 環境変数を利用して、参照する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
```

付録 **A**

日本語専用ライブラリ (libjapanese.a)

JFP では、日本語固有の処理を行うために、`libjapanese.a` [1] が提供されています。この付録では、次の項目について説明します。

注 - `libjapanese.a` と、これに関連する次のヘッダーファイルは、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、ソースファイルを使って、ソースの互換性を保つための代替関数とマクロを提供します。 詳細は、SUNWjlibj パッケージに含まれる（Entire インストールにのみ含まれる） `/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 japanese ロケールに設定されていることを前提としていま

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。

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併用し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) 関数が呼び出され、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 とも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 -ljapaneses 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 5.8 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 widec.h ではなく、/usr/include/wstring.h を取り込んでください。表 A–1 に、ワイド文字列処理関数 (マクロ) を示します。

表 A–1 ワイド文字列処理関数

関数	機能
wstrcat()	文字列を追加する
wstrncat()	指定文字数分だけ文字列を追加する
wstrcmp()	文字列同士の比較を行う
wstrncmp()	指定文字数分だけ文字列同士の比較を行う

表 A–1 ワイド文字列処理関数 続く

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

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
wstrtol()	主および補助コードセット文字列を long 整数に変換する
wstrtod()	主および補助コードセット文字列を倍精度に変換する

例 A–1 は、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です。このプログラム

では、getws(3C) 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。

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 = ''0123456789０１２３４５６７８９'';
 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 = wstrtol(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
３４５３４５　３４５３４５
３９８５７　３９８　５８３４５８９
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は、単一バイト変換関数 (toupper、tolower) と同様の変換関数を 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する
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する
tojhira()	JIS X 0208 で定義されるカタカナ文字、カタカナ繰り返し記号を表すワイド文字を、対応するひらがな文字、ひらがな繰り返し記号を表すワイド文字に変換する
tojkata()	JIS X 0208 で定義されるひらがな文字、ひらがな繰り返し記号を表すワイド文字を、対応するカタカナ文字、カタカナ繰り返し記号を表すワイド文字に変換する
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する
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する
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 型に変換する
kutentojis()	区点番号からワイド文字に変換する

例 A–2 は、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です。このプログラム例では、getws(3C)

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

例 **A–2**　ワイド文字変換用関数

```
sun% cat 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_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。
「ヴヵヶ」ハ、変換サレマセン。
クリカエシキゴウ - ゞヽ
sun% cc -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
さんぷるぷろぐらむです。
「ヴヵヶ」は、変換されません。
くりかえしきごう - ゞヽ
sun%
```

ワイド文字分類関数 (jctype)

日本語専用ライブラリは、文字の分類機能を提供します。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
`isjis()`	JIS X 0208 で定義される文字を表すワイド文字のとき、真を返す
`isjalpha()`	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、真を返す
`isjupper()`	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、真を返す
`isjlower()`	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、真を返す
`isjdigit()`	JIS X 0208 で定義される数字を表すワイド文字のとき、真を返す
`isjspace()`	JIS X 0208 で定義される空白文字を表すワイド文字のとき、真を返す
`isjpunct()`	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、真を返す
`isjparen()`	JIS X 0208 で定義される括弧記号を表すワイド文字のとき、真を返す
`isjline()`	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、真を返す
`isjunit()`	JIS X 0208 で定義される単位記号を表すワイド文字のとき、真を返す
`isjsci()`	JIS X 0208 で定義される学術記号を表すワイド文字のとき、真を返す
`isjgen()`	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、真を返す
`isjkanji()`	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、真を返す
`isjspecial()`	JIS X 0208 で定義される特殊文字を表すワイド文字のとき、真を返す
`isjgreek()`	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、真を返す
`isjrussian()`	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、真を返す

表 **A–4** ワイド文字分類関数 続く

関数	機能
isjkata()	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、真を返す
isjhira()	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、真を返す
isj1bytekana()	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、真を返す
isjhankana()	isj1bytekana() 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します。このプログラム例では、getws(3C) 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) 関数を使って ASCII に変換しています。

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%
```

コード変換用関数 (jisconv、ibmjcode)

日本語専用ライブラリは、JIS X 0208 文字集合のコード変換機能を提供します。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) 間でコード変換を行います。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 の漢字セットと 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 漢字文字に変換する
cjistouj()	漢字の JIS 文字 1 文字を日本語 EUC コード文字に変換する
cujtojis()	漢字の日本語 EUC コード文字 1 文字を JIS 文字に変換する
cujtosj()	漢字の日本語 EUC コード文字 1 文字を PC 漢字文字に変換する
csjtojis()	PC 漢字文字 1 文字を JIS 文字に変換する
csjtouj()	PC 漢字文字 1 文字を日本語 EUC コード文字に変換する
jis7tosj()	7 ビット JIS 文字の文字列を PC 漢字文字の文字列に変換する

表 **A–5** コード変換用関数 続く

関数	機能
jis7touj()	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する
sjtojis7()	PC 漢字文字の文字列を 7 ビット JIS 文字の文字列に変換する
ujtojis7()	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する
jis8tosj()	8 ビット JIS 文字の文字列を PC 漢字文字の文字列に変換する
jis8touj()	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する
sjtojis8()	PC 漢字文字の文字列を 8 ビット JIS 文字の文字列に変換する
ujtojis8()	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する
jistosj()	jis8tosj() と同じ
jistouj()	jis8touj() と同じ
ujtojis()	ujtojis8() と同じ
ujtosj()	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する
sjtojis()	sjtojis8() と同じ
sjtouj()	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する

例 A–4 は、このコード変換用関数を使って、EUC から JIS コードへコード変換するプログラム例です。

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 | jistoeuc
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%
```

IBM 漢字コードと日本語 EUC コード間のコード変換を行う一連の関数も提供されています (ibmjcode(3X) 参照)。これらの関数を使用する場合は、/usr/include/ibmjcode.h を取り込んでください。

付録 **B**

Solaris の日本語 TrueType フォント

この付録では、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。表 B–1 に、サポートしているフォントの Platform ID と Encoding ID を示します。

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)

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

Sorais 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

ズのグリフを表示するとき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提供しています。

- 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。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 と組み合わせて使用します。

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
```

Soralis 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 フォントと 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)、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使ったアプリケーションとの互換性を保つため、リコーのフォントを使用するように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 を採用しています。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	

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™ プログラムの動作は保障されません。Solaris 2.5.1 以前にフォントマネージャ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)` やカレンダマネージャで出力された PostScript は、DPS では表示できません。

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は、以下の手順に従ってください。

1. **TrueType フォントを適当なディレクトリにコピーします。なお、Solaris では TrueType Collection (.ttc) フォントを直接は使用できません。**

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

フォントの置き場所は、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) のマニュアルページを参照してください。フォントの **XLFD (X Logical Font Description)** 名は、**PIXEL_SIZE**、**POINT_SIZE**、**RESOLUTION_X**、**RESOLUTION_Y**、**AVERAGE_WIDTH** の各フィールドではすべて **0** にします。また **SPACING** フィールド 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'** にします。 **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** 補助漢字の一部のグリフがサポートされていません。使用できるのは、**Microsoft** 標準キャラクタセットの **IBM** 拡張文字にある一部のグリフのみです。

   - 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 : 一時的に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な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なった場合の動作は保証されません。

索引

X

か

け

こ

し

て

は

ひ

ふ

ほ

め

も

わ