ICS 35.100.01
L 74

中华人民共和国国家标准

GB/T 16681—1996

信息技术　开放系统中文界面规范

Information technology
Chinese interface specifications of open system

1996-12-18发布　　1997-07-01实施

国家技术监督局 发布

目　　次

前　言

本标准的附录A是标准的附录，附录B和附录C是提示的附录。

本标准由中华人民共和国电子工业部提出。

本标准由电子工业部标准化研究所归口。

本标准起草单位：中国计算机软件与技术服务总公司、中国科学院软件研究所、联想集团公司、信息高技术联合研究所。

本标准主要起草人：孟庆余、孙龙驹、姚吉松、吴健、姚建枫、慈林林、赵国宏、戴世宗、冯惠。

引　　言

本标准是为基于POSIX或XPG(见附录C)的中文处理系统规定的基本界面规范。本标准的制定将对系统中文平台的开发、应用、移植、互换起重要指导作用。

本标准的主要内容包括：国际化要求、本地化要求、I/O服务功能界面、实用程序要求、图形界面要求等。

中华人民共和国国家标准

信息技术　开放系统中文界面规范　GB/T 16681—1996

Information technology
Chinese interface specifications of open system

1　范围

1.1　主题内容

本标准定义开放系统中文平台应用程序编程界面、I/O 服务功能界面以及图形界面。

1.2　适用范围

本标准适用于开放系统中文平台的开发、应用和基于该环境的中文处理软件的集成。

2　引用标准

下列标准包含的条文，通过在本标准中引用而构成为本标准的条文。本标准出版时，所示版本均为有效。所有标准都会被修订，使用本标准的各方应探讨使用下列标准最新版本的可能性。

GB 1988—89	信息处理　信息交换用七位编码字符集 (eqv ISO 646:1983)
GB 2312—80	信息交换用汉字编码字符集　基本集
GB/T 5261—94	信息处理　七位和八位编码字符集用的控制功能　(idt ISO 6429:1988)
GB/T 7408—94	数据元和交换格式　信息交换　日期和时间表示法(eqv ISO 8601:1988)
GB 13000.1—93	信息技术　通用多八位编码字符集(UCS)　第一部分：体系结构与基本多文种平面　(idt ISO/IEC 10646-1:1993)
GB/T 13846—92	图形信息交换用矢量汉字　仿宋体字模集及数据集
GB/T 13847—92	图形信息交换用矢量汉字　楷体字模集及数据集
GB/T 13848—92	图形信息交换用矢量汉字　黑体字模集及数据集
GB/T 14246.1—93	信息技术　可移植操作系统界面　第一部分：系统应用程序界面(POSIX.1)(idt ISO/IEC 9945-1:1990)
GB/T 15272—94	程序设计语言 C　(idt ISO/IEC 9899:1990)
ISO/IEC 9945-2:1993	信息技术　可移植操作系统界面　第二部分：命令和实用程序(POSIX.2)

3　定义

本标准采用下列定义。

3.1　多字节字符　multibyte character

用若干个字节表示的字符。

3.2　单字节字符　single-byte character

用一个字节表示的字符。

3.3　宽字符　wide character

国家技术监督局 1996-12-18 批准　　1997-07-01 实施

为了统一地处理单字节字符与多字节字符，而采用的具有统一编码宽度的字符的内部表示。

3.4 字符宽度 character width

字符所占用的存储单位数或显示列位置数。字符宽度有显示宽度与存储宽度之分，而且字符的显示宽度与存储宽度不总是一致的。

3.5 输入法 input method

为输入特定字符集中字符而定义的方法。

3.6 输入提示屏 input prompt screen

计算机屏幕上输入法要求的显示区。

3.7 热键 hot key

计算机键盘上输入法要求的方式切换键。

3.8 点阵汉字 dot-matrix Chinese character

用位映象图描述的汉字字符。

3.9 矢量轮廓汉字 vector outline Chinese character

用有向线段描述汉字轮廓的汉字字符。

3.10 曲线轮廓汉字 curve outline Chinese character

用特定曲线描述汉字轮廓的汉字字符。

4 要求

4.1 总体要求

4.1.1 标准的符合性

具体实现应遵循本标准；此外，还应符合操作系统的其他标准。

凡本标准中未明确定义的符号，请参阅相应的标准。

4.1.2 C 语言约束

本标准使用按 GB/T 15272 描述编程界面，并不限定在具体实现时所使用的编程语言。

4.1.3 国际化/本地化(I18N/L10N)模式

具体实现应采用 POSIX 标准的国际化/本地化模式。

4.1.4 字处理原则

在系统字处理中，必须以字符为单位，而不能以字节为单位。若无特殊说明，本标准中所用的“字符”既可以是单字节字符也可以是多字节字符。

4.1.5 前导文件

若无特殊说明，本标准所引用的前导文件与 XPG4(见附录 C)一致。

4.2 国际化

国际化(Internationalization)是规定在一个计算机程序内部的能力，使它适应于不同的本地语言、本地风俗和编码字符集。

本标准定义一组国际化的应用程序编程界面。

由于多字节字符事实上的存在，并表现为不等长，为了处理上的一致性、规范性，字符处理时应以等长的宽字符表示多字节字符。

4.2.1 标准 C 多字节功能

a) **名称**

mbtowc——把一个字符转换成宽字符。

格式

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc,const char *s,size_t n);
```

说明

如果 s 不是空指针，那么 mbtowc()先确定由 s 指向的字符的字节数，然后确定与该字符相对应的 wchar_t 型的宽字符代码值(与空字符等价的宽字符代码值是零)。如果该字符是有效的并且 pwc 不是空指针，mbtowc()把宽字符代码值存放在 pwc 指向的目标中。

本函数受当前本地环境 LC_CTYPE 影响。对于依赖于状态的编码，通过将参数 s 置为空来调用本函数，使函数设置为初始状态；后续的调用，如果参数 s 非空，则使函数的初始状态根据需要改变。如果编码依赖于状态，那么参数 s 为空指针的调用使本函数返回一个非零值，否则返回零。如果在实现中使用一些特殊字节来改变切换状态，那么这些字节不生成独立的宽字符代码，而是与一个相邻的字符组合在一起。改变当前的 LC_CTYPE 使本函数的初始状态不确定。s 指向的数组中至多 n 个字节被检查。

返回

当 s 是空指针时，如果字符编码依赖于状态，那么函数 mbtowc()返回一个非零值；如果编码不依赖于状态，则函数返回零值。当 s 是非空指针时，如果 s 指向空字节，那么 mbtowc()返回零；或者如果 n 个或少于 n 个字节组成了合法的多字节字符，则返回多字节字符的字节数；或者如果是无效字符，则函数返回−1 并置 errno 以示出错。

不可能返回大于 n 或者宏 MB_CUR_MAX 的值。

b) **名称**

mblen——获取一字符的字节数。

格式

```
#include <stdlib.h>
int mblen(const char *s,size_t n);
```

说明

如果 s 不是空指针，mblen()确定由 s 指向的字符的字节数。除了 mbtowc 的切换状态不受影响之外，它等价于：

```
mbtowc((wchar_t *)0,s,n);
```

本函数受当前本地环境 LC_CTYPE 影响。对于依赖于状态的编码，通过将参数 s 置为空来调用本函数，使函数设置为初始状态；后续的调用，如果参数 s 非空，则使函数的初始状态根据需要改变。如果编码依赖于状态，那么参数 s 为空指针的调用使本函数返回一个非零值，否则返回零。如果在实现中使用一些特殊字节来改变切换状态，那么这些字节不生成独立的宽字符代码，而是与一个相邻的字符组合在一起。改变当前的 LC_CTYPE 使本函数的初始状态不确定。

返回

当 s 是空指针时，如果字符编码依赖于状态，那么函数 mblen()返回一个非零值；如果编码不依赖于状态，则函数返回零值。当 s 是非空指针时，如果 s 指向空字节，那么 mblen()返回零；或者如果 n 个或少于 n 个字节组成了合法的多字节字符，则返回多字节字符的字节数；或者如果是无效字符，则函数返回−1 并置 errno 以示出错。

不可能返回大于 n 或者宏 MB_CUR_MAX 的值。

c) **名称**

wctomb——把宽字符转换成一个字符。

格式

```
#include<stdlib.h>
int wctomb(char *s,wchar_t wchar);
```

说明

wctomb()确定 wchar 表示的宽字符(包括切换状态的变化)所对应的字符的字节数,并把对应的字符(可能是多字节和改变切换状态的特殊字节)存放在 s(如果 s 不是空指针)指向的目标数组中。最多存放 MB _CUR _MAX 个字节。如果 wchar 的值为零,wctomb()被置为初始切换状态。

本函数受当前本地环境 LC _CTYPE 影响。对于依赖于状态的编码,通过将参数 s 置为空来调用本函数,使函数设置为初始状态;后续的调用,如果参数 s 非空,则使函数的初始状态根据需要改变。如果编码依赖于状态,那么参数 s 为空指针的调用使本函数返回一个非零值,否则返回零。如果在实现中使用一些特殊字节来改变切换状态,那么这些字节不生成独立的宽字符代码,而是与一个相邻的字符组合在一起。改变当前的 LC _CTYPE 使本函数的初始状态不确定。

返回

当 s 是空指针时,如果字符编码依赖于状态,那么函数 wctomb()返回一个非零值;如果编码不依赖于状态,则函数返回零值。当 s 是非空指针时,如果 wchar 不对应合法的字符,那么函数返回−1;否则返回 wchar 对应的字节数。
不可能返回大于宏 MB _CUR _MAX 的值。

d) **名称**

mbstowcs——把一个字符串转换成一宽字符串。

格式

```
#include<stdlib.h>
size_t mbstowcs(wchar_t * pwcs,const char *s,size_t n);
```

说明

mbstowcs()以初始切换状态开始,把 s 指向的字符序列转换成对应的宽字符代码序列,并把不多于 n 个的宽字符代码存放在 pwcs 指向的数组中。空符(空符将被转换成一个值为零的宽字符代码)后续的字符不会被检查和转换。除了 mbtowc()的切换状态不受影响外,每个字符的转换就象调用 mbtowc()一样。

pwcs 指向的数组中最多存储 n 个元素。如果拷贝目标重叠,其行为没被定义。

本函数受当前本地环境的 LC _CTYPE 影响。如果 pwcs 是一空指针,mbstowcs()返回宽字符数组所需的元素数。

返回

如果遇到一无效字符,mbstowcs()返回(size _t)−1,并且置 errno 以示出错;否则 mbstowcs()返回被转换的数组元素的个数(或者如果 pwcs 是空指针,则是所需的个数),不包括作为终止符的零代码。如果返回值是 n,数组不以零代码终止。

e) **名称**

wcstombs——把一宽字符串转换成一字符串。

格式

```
#include<stdlib.h>
size_t wcstombs(char *s,const wchar_t *pwcs,size_t n);
```

说明

wcstombs()以初始切换状态开始,把 pwcs 指向的数组中的宽字符代码序列转换成字符

序列，并把这些字符存放在 s 指向的数组中，当超过总共 n 个字节的限制或者存放了一个空字节时转换结束。除了 wctomb()的切换状态不受影响外，每个宽字符的转换就象调用 wctomb()一样。

本函数受当前本地环境的 LC _CTYPE 影响。

s 指向的数组中最多存储 n 个字节。如果拷贝目标重叠，其行为没被定义。如果 s 是一个空指针，wcstombs()返回字符数组所需的字节数。

返回

如果遇到一宽字符代码，它不对应有效字符(单字节或多字节)，wcstombs()返回(size _t)—1；否则，wcstombs()返回字符数组中的字节数，不包括作为终止符的空字节。如果返回值是 n，数组不以空终止。

4.2.2 I/O 功能

a) **名称**

fgetwc——从一个流中获取宽字符代码。

格式

```
#include<stdio.h>
#include<wchar.h>
wint _t fgetwc(FILE *stream);
```

说明

fgetwc()从 stream 指向的输入流中获取下一个字符(如果存在)，再将该字符转换成对应的宽字符代码，并且下移流的文件位置指示器(如果已定义)。

如果出现错误，流的文件位置指示器的值不确定。

fgetwc()可以因为更新而标记 stream 所指文件的 st _atime 域。如果 fgetc()、fgetwc()、fgetws()、fread()、fscanf()、getc()、getchar()、gets()或 scanf()第一次成功地从 stream 中获取的数据不是前面 ungetc()或者 ungetwc()的调用提供的，则因为更新而标记 st _atime 域。

返回

fgetwc()的成功调用将从 stream 指向的输入流中读取字符并返回对应的 wint _t 型的宽字符代码。如果流遇到了文件尾，流的文件结束指示器将被设置并且 fgetwc()返回 WEOF。如果发生读错误，流的出错信息指示器将被设置，fgetwc()返回 WEOF 并且置 errno 以示出错。

b) **名称**

getwc——从一个流中获取宽字符。

格式

```
#include<stdlib.h>
#include<wchar.h>
wint _t getwc (FILE *stream);
```

说明

getwc()等价于 fgetwc()，但它作为宏实现时略有不同。

返回

见 fgetwc()。

c) 名称

getwchar——从标准输入中获取宽字符。

格式

```
#include<wchar.h>
wint_t getwchar (void);
```

说明

getwchar()的功能等价于 getwc(stdin)。

返回

见 fgetwc()。

d) 名称

fgetws——从一个流中获取一宽字符串。

格式

```
#include<stdio.h>
#include<wchar.h>
wchar_t * fgetws(wchar_t * ws,int n,FILE * stream);
```

说明

fgetws()从 stream 中不断读取字符并转换成对应的宽字符代码,把结果存放在 ws 指向的 wchar_t 型数组中,直到读取了 n-1 个字符,或者是遇到换行符(换行符被转换并存放到 ws),或者文件尾为止。这样 ws 以宽字符的空符结束。

如果出现错误,流的文件位置指示器的值将不确定。

fgetws()可以因为更新而标记 stream 所指文件的 st_atime 域。如果 fgetc()、fgetwc()、fgetws()、fread()、fscanf()、getc()、getchar()、gets()或 scanf()第一次成功地从 stream 中获取的数据不是前面 ungetc()或者 ungetwc()的调用提供的,则因为更新而标记 st_atime 域。

返回

fgetws()的成功调用返回 ws。如果遇到文件尾,流的文件结束指示器将被设置并且 fgetws()返回空指针。如果发生读错误,流的出错信息指示器被设置,fgetws()返回空指针并且置 errno 以示出错。

e) 名称

fputwc——在一个流上输出宽字符代码。

格式

```
#include <stdio.h>
#include <wchar.h>
wint_t fputwc (wint_t wc,FILE * stream);
```

说明

fputwc()把与宽字符代码 wc 对应的字符写到 stream 指向的输出流中,并且下移流的文件位置指示器(如果已定义)。如果文件不能支持定位请求,或者流是以附加方式打开的,字符将附加到输出流后。当写字符出现错误时,输出文件的转换状态将是无定义状态。

在对同一流的 fputwc()与下次 fflush()或 fclose()的成功调用之间,或者直到调用 exit()

或 abort()为止，将因为更新而标记文件的 st _ ctime 和 st _ mtime 域。

返回

如果成功，fputwc()返回 wc。否则返回 WEOF，流的出错信息指示器被设置，并且 置errno 以示出错。

f) **名称**

putwc——把一宽字符输出到一个流上。

格式

```
#include <stdio.h>
#include <wchar.h>
wint _ t putwc (wint _ t wc,FILE * stream);
```

说明

putwc()等价于 fputwc()，但它作为宏实现略有不同。

返回

见 fputwc()。

g) **名称**

putwchar——把宽字符写到标准输出。

格式

```
#include <wchar.h>
wint _ t putwchar (wintr _ t wc);
```

说明

putwchar(wc)等价于 putwc(wc,stdout)。

返回

见 fputwc()。

h) **名称**

fputws——在一个流上输出宽字符串。

格式

```
#include <stdio.h>
#include <wchar.h>
int fputws (const wchar _ t * ws,FILE * stream);
```

说明

fputws()把与 ws 指向的(以空结束的)宽字符串对应的字符串写入 stream 指向的流中。与空宽字符代码相对应的字符不被写入。在对同一流的 fputwc()与下次 fflush()或 fclose()的成功调用之间，或者直到调用 exit()或 abort()为止，将因为更新而标记文件的 st _ ctime 和 st _ mtime 域。

返回

如果成功，fputws()返回一非负整数；否则返回-1，置流的出错信息指示器并且置 errno 以示出错。

i) **名称**

ungetwc——把一宽字符代码放回输入流。

格式

```
#include <stdio.h>
#include <wchar.h>
wint _ t ungetwc (wint _ t wc,FILE * stream);
```

说明

ungetwc()把与宽字符代码 wc 相对应的字符放回 stream 指向的输入流。被放回的字符将在下次从流中读时返回。一个文件定位函数(fseek()、fsetpos()或 rewind())的成功调用会清除放回流(stream 所指向的流)中的字符。与流对应的外部存储将不做改变。

ungetwc()的一次调用可保证字符的放回。如果在同一个流中太多次调用 ungetwc(),而没有读文件或者文件定位操作来间隔,操作可能失败。

如果 wc 的值等于宏 WEOF 的值,操作失败并且输入流不做改变。

ungetwc()的成功调用将清除流的文件结束指示器。在读操作或者清除所有放回字符之后的文件位置指示器的值与放回字符前是一样的。每一次 ungetwc()的成功调用将使文件位置指示器减小(减小 1 或更多),如果在调用之前它的值是零,在调用之后它的值将不确定。

返回

如果成功,ungetwc()返回与放回字符相对应的宽字符代码值,否则它返回 WEOF。

4.2.3 串操作功能

a) **名称**

wcscat——连接两个宽字符串。

格式

```
#include <wchar.h>
wchar_t * wcscat (wchar_t * ws1,const wchar_t * wsb);
```

说明

wcscat()把 ws2 指向的宽字符串的副本(包括宽字符代码结束符)附加到 ws1 所指的宽字符串的末尾。ws2 的起始宽字符盖写了 ws1 末尾的宽字符空符。如果拷贝目标重叠,该行为未定义。

返回

wcscat()返回串 ws1;不规定发生错误的返回值。

b) **名称**

wcsncat——部分连接两个宽字符串。

格式

```
#include <wchar.h>
wchar_t * wcsncat (wchar_t * ws1,const wchar_t * ws2,size_t n);
```

说明

wcsncat()把 ws2 指向的数组中的最多 n 个宽字符代码附加(空宽字符及其后面的宽字符代码不被附加)到 ws1 所指的宽字符串的末尾。ws2 的起始宽字符盖写了 ws1 末尾的宽字符空符。通常一个宽字符结束符要附加到结果中。如果拷贝目标重叠,该行为未定义。

返回

wcsncat()返回串 ws1;不规定发生错误的返回值。

c) **名称**

wcscmp——比较两个宽字符串。

格式

```
#include <wchar.h>
```

int wcscmp (const wchar_t * ws1,const wchar_t * wsb);

说明

wcscmp()比较 ws1 指向的宽字符串和 ws2 指向的宽字符串。

返回的非零值由被比较的目标中的第一处不同值决定。

返回

wcscmp()比较完成后，根据 ws1 指向的字符大于，等于或小于 ws2 指向的字符，返回一大于、等于或小于零的整数。

d) **名称**

wcsncmp——部分比较两个宽字符串。

格式

#include <wchar.h>

int wcsncmp (const wchar_t * ws1,const wchar_t * ws2,size_t n);

说明

wcsncmp()比较 ws1 指向的宽字符串和 ws2 指向的宽字符串中最多 n 个宽字符代码，(空宽字符及其后面的宽字符代码不被比较)。

返回的非零值由被比较的目标中的第一处不同值决定。

返回

wcsncmp()比较完成后，根据 ws1 指向的字符大于，等于或小于 ws2 指向的字符，返回一大于、等于或小于零的整数。

e) **名称**

wcscpy——复制一宽字符串。

格式

#include <wchar.h>

wchar_t * wcscpy (wchar_t * ws1,const wchar_t * wsb);

说明

wcscpy()把 ws2 指向的宽字符串(包括宽字符代码的结束符)复制到 ws1 所指的数组中，如果拷贝目标重叠，该行为未定义。

返回

wcscpy()返回串 ws1；不规定发生错误的返回值。

f) **名称**

wcsncpy——部分复制一宽字符串。

格式

#include <wchar.h>

wchar_t * wcsncpy (wchar_t * ws1,const wchar_t * ws2,size_t n);

说明

wcsncpy()把 ws2 指向的宽字符串中最多 n 个字符复制到 ws1 所指的数组中(空宽字符及其后面的宽字符代码不被复制)。如果拷贝目标重叠，该行为未定义。

返回

wcsncpy()返回串 ws1；不规定发生错误的返回值。

g) **名称**

wcslen——得到宽字符串的长度。

格式

```
#include <wchar.h>
size_t wcslen (const wchar_t * ws);
```

说明

wcslen()计算 ws 指向的宽字符串中宽字符代码个数，不包括宽字符结束符在内。

返回

wcslen()返回 ws 中的宽字符数；不规定发生错误的返回值。

h) **名称**

wcschr——宽字符串扫描操作

格式

```
#include <wchar.h>
wchar_t * wcschr (const wchar_t * ws,wint_t wc);
```

说明

wcschr()确定 ws 所指向的宽字符串中第一次出现 wc 的位置。wc 必须是可说明为 wchar_t 型的字符并且是在当前本地环境中与一有效字符相对应的宽字符。宽字符结束符被认为是宽字符串的一部分。

返回

如果成功，wcschr()返回一指向宽字符代码的指针；否则，如果 wc 没有在该宽字符串中出现，返回一空指针。

i) **名称**

wcsrchr——宽字符串扫描操作。

格式

```
#include <wchar.h>
wchar_t * wcsrchr (const wchar_t * ws,wint_t wc);
```

说明

wcsrchr()确定 ws 所指向的宽字符串中最后一次出现 wc 的位置。wc 必须是可说明为 wchar_t 型的字符并且是在当前本地环境中与一有效字符相对应的宽字符。宽字符结束符被认为是宽字符串的一部分。

返回

如果成功，wcsrchr()返回一指向宽字符代码的指针；否则，如果 wc 没有在宽字符串中出现，返回一空指针。

j) **名称**

wcspbrk——在宽字符串中扫描宽字符代码。

格式

```
#include <wchar.h>
wchar_t * wcspbrk (const wchar_t * ws1,const wchar_t * wsb);
```

说明

wcspbrk()确定 ws2 所指的宽字符串中任一宽字符代码在 ws1 所指的宽字符串中第一次出现的位置。

返回

如果成功，wcspbrk()返回一指向宽字符代码的指针；否则，如果 ws2 中的宽字符没有在 ws1 出现，返回一空指针。

k) **名称**

wcsspn——获取宽字符子串的长度。

格式

#include <wchar.h>

size_t wcsspn (const wchar_t * ws1,const wchar_t * wsb);

说明

wcsspn()计算 ws1 指向的宽字符串的初始段最大长度，该段完全由 ws2 所指宽字符串中的代码组成。

返回

wcsspn()返回串 ws1 中初始段的长度；不规定发生错误的返回值。

l) **名称**

wcscspn——获取宽字符子串的长度。

格式

#include <wchar.h>

size_t wcscspn (const wchar_t * ws1,const wchar_t * wsb);

说明

wcscspn()计算 ws1 指向的宽字符串的初始段最大长度，该段完全由不是 ws2 所指宽字符串中的代码组成。

返回

wcscspn()返回串 ws1 中初始段的长度；不规定发生错误的返回值。

m) **名称**

wcstok——把宽字符串分裂成记号。

格式

#include <wchar.h>

wchar_t * wcstok (wchar_t * ws1,const wchar_t * wsb);

说明

wcstok()的连续调用将把 ws1 所指向的宽字符串分隔成一记号序列，每一个被 ws2 中的宽字符分隔。在连续调用中，第一次调用把 ws1 作为函数的第一个参数，随后的调用用空指针作为它的第一个参数。每次调用 ws2 所指向的分隔符串可以不同。

在连续调用中，第一次调用在 ws1 所指的宽字符串中搜索第一个不包括在 ws2 当前所指的分隔符串中的宽字符代码。如果没找到这样的宽字符代码，即 ws1 所指的宽字符串中没有记号，则 wcstok()返回一空指针。如果找到了，它便是第一个记号的开始。

wcstok()从记号的开始处起搜寻当前分隔符串中包含的宽字符代码。如果没有找到这样的宽字符代码，则当前记号延伸至 ws1 所指的宽字符串的结尾，并且随后的搜寻将返回空指针。如果找到了这样的宽字符，它将被一个空宽字符盖写，该空宽字符结束当前记号。wcstok()保存指向后续宽字符代码的指针，以便下次搜寻从这里开始。

随后的每一次调用，用空指针作为第一个参数，从保存的指针开始并且按照上述过程继续搜寻。

返回

如果成功，wcstok()返回指向一个记号的第一个宽字符代码的指针；否则，如果没有记号，wcstok()返回一空指针。

n) **名称**

wcswcs——寻找宽字符子串。

格式

```
#include <wchar.h>
wchar_t * wcswcs (const wchar_t * ws1,const wchar_t * wsb);
```

说明

wcswcs()确定 ws2 所指的宽字符串中的宽字符序列(不包括结尾的空宽字符)在 ws1 所指的宽字符串中第一次出现的位置。

返回

如果成功,wcswcs()返回确定的宽字符串指针。否则,如果宽字符串没被找到,返回一空指针。

如果 ws2 指向一零宽的宽字符串,函数返回 ws1。

4.2.4 打印功能

a) **名称**

fprintf、printf、sprintf——打印格式输出。

格式

```
#include <stdio.h>
int fprintf (FILE * stream,const char * format,...);
int printf (const char * format,...);
int sprintf (char * s,const char * format,...);
```

说明

注:此处只说明有关中文处理的部分。

fprintf()输出到命名的输出流中;printf()输出到标准输出 stdout 中;sprintf()输出到 s 中。每个函数按照 format 转换格式化打印。format 是一个以初始转换状态开始和结束的字符串,format 由零或多个指令组成:普通字符,它将被拷贝到输出流;转换说明符,每个都将匹配零或多个参数。如果没有足够的参数与 format 匹配,结果未定义。如果参数超过格式转换所需要的数目,多余的参数被求值或被忽略。

%n$将代替转换说明符%,n 是[1,(NL_ARGMAX)]中的一个十进制整数,它给出参数序列中参数的位置。转换说明符适用于 format 后面的参数序列中的 n 个参数,而不适用更后面的没有用到的参数。这种特点为特定语言按次序选择参数提供了格式定义。

格式串中包含%n$形式的转换说明符,参数序列中编号的参数可以根据需要重复使用。

在一个包含%n$转换说明符的格式串中,一个域的宽度和精确度由 *m$ 指定,m 是[1,(NL_ARGMAX)]中的一个十进制整数,它给出参数在参数序列中的位置,该参数表示一整参数的域宽或精确度。例如:

```
printf("%1$d:%2$.*3$d:%4$.*3$d[BS)]",hour,min,precision,sec);
```

format 中可以含有编号参数说明符(即%n$和*m$),或者无编号参数说明符(即%和*),但通常不同时含有二者,只有一个例外%%可以与%n$形式混用。编号参数说明符

与无编号参数说明符在一 format 中混用没被定义。当用编号参数说明时，如果说明第 n 个参数，则需要从第 1 到第 n—1 个前导参数都已在格式串中被说明。

标志符及其意义：

' 十进制转换(%i,%d,%u,%f,%g 或%G)的整数部分用千分号格式化。其他转换没被定义。使用了非货币分组符号。

转换符及其意义：

C wchar _ t 型的参数被转换成表示一个字符的字节序列，并且写出结果字符。如果指定了精确度，则结果没有定义。转换类似于 wctomb()。在一个本地环境，如果编码依赖于状态，则与流的切换状态有关的行为依赖于具体的实现。

S 参数必须是 wchar _ t 型的数组指针。数组中以空宽字符结束但不包括结束空宽字符的宽字符代码序列被转换成一字节序列，并且写出结果字节。如果指明了精确度，则不超出限制数的字节和完整的字符被写出。如果没有指明精确度，或者精确度大于转换的字节数组的大小，那么宽字符数组必须用一空宽字符结束。转换类似于 wcstombs()。在一个本地环境，如果编码依赖于状态，则与流的切换状态有关的行为依赖于具体的实现。

返回

如果成功，返回被传送的字节数，sprintf()的返回值不包括结束空符；如果输出发生错误，则返回一负值。

4.2.5 查询区域环境信息

a) **名称**

localeconv——确定程序本地环境。

格式

```
#include <locale.h>
struct lconv * localeconv (void);
```

说明

localeconv()根据当前本地环境的规则用适合于格式化的数字量(货币和其他)来设置 struct lconv 结构中的成员。

结构中 char * 型的成员是指向串的指针，其中任何一个(decimal _ point 除外)都可以指向""，以示该值在当前本地环境不可用或者是长度为零。char 型的成员是非负的数值，其中任何一个可以是{CHAR _ MAX}，以示该值在当前本地环境不可用。

包括的成员如下：

注：此处只说明中文处理需要增加的部分。

char p _ cs _ precedes 如果 currency _ symbol 或 int _ curr _ symbol 在非负的格式化过的货币数值前面，则置为 1；如果在其后面，则置为 0。

char p _ sep _ by _ space 如果没有空格把符号 currency _ symbol 或 int _ curr _ sym _ bol

与一非负的格式化的货币数值分隔开，则置为 0。如果一个空格把符号与数值分隔开了，则置为 1；如果一个空格分隔了符号和邻近的符号串，则置为 2。

char n _ cs _ precedes　如果 currency _ symbol 或 int _ curr _ symbol 在负的格式化过的货币数值前面，则置为 1；如果在其后面，则置为 0。

char n _ sep _ by _ space　如果没有空格把符号 currency _ symbol 或 int _ curr _ symbol 与一负的格式化的货币数值分隔开，则置为 0。如果一个空格把符号与数值分隔开了，则置为 1；如果一个空格分隔了符号和邻近的符号串，则置为 2。

p _ sign _ posn 和 n _ sign _ posn 的值说明如下：

0　圆括号括起数量值和 currency _ symbol 或者 int _ curr _ symbol。

1　符号串在数量值和 currency _ symbol 或者 int _ curr _ symbol 的前面。

2　符号串在数量值和 currency _ symbol 或者 int _ curr _ symbol 的后面。

3　符号串直接在 currency _ symbol 或者 int _ curr _ symbol 的前面。

4　符号串直接在 currency _ symbol 或者 int _ curr _ symbol 的后面。

返回

localeconv()返回一指向被填充的目标的指针。返回值指向的结构不可以被程序修改，但可以被 localeconv()的后续调用盖写。另外，如果用 LC _ ALL、LC _ MONE TARY 或者 LC _ NUMERIC 来调用 setlocale()可以盖写结构中的内容。

b) **名称**

nl _ langinfo——语言信息。

格式

```
#include <langinfo.h>
char * nl _ langinfo (nl _ item item);
```

说明

nl _ langinfo()返回一字符串指针，该串中包含与程序的本地环境中定义的特殊语言或文化区域有关的信息。说明的常量名和 item 的值在<langinfo.h>中定义。例如：

nl _ langinfo(ABDAY _ a)

如果标识的语言是葡萄牙语，它将返回一个指向"Dom"的串指针；而如果标识的语言是英语，则返回"Sun"。

返回

如果在一个本地环境中没有定义 langinfo 数据，则 nl _ langinfo()返回 POSIX 本地环境中相应的指针。在所有本地环境中，如果 item 包括一无效设置，则 nl _ langinfo()返回指向一空串的指针。

c) **名称**

strfmon——把货币值转换成串。

格式

```
#include <monetary.h>
size _ t strfmon (char *s,size _ t maxsize,const char * format,...);
```

说明

strfmon 把由 format 指向的串所控制的字符放入 s 指向的数组中，多于 maxsize 的字节不放入数组中。

格式是一个字符串，它包含两种类型的目标：简单拷贝到输出流的普通字符和转换说明，每一个转换说明都使零个或多个参数被转换和格式化。如果没有足够的参数用于格式化，结果没被定义。如果格式化结束，而参数多余，则多余的参数被忽略。

返回

如果结果的字节总数包括结尾的空字节不超过 maxsize，则 strfmon()返回 s 指向的数组中放入的字节数，不包括结尾的空字节。否则，返回－1，数组的内容不确定，并且置 error 以示出错。

4.2.6 日期和时间

a) **名称**

strftime——把日期和时间转换成串

格式

```
#include <time.h>
size_t strftime (char *s,size_t maxsize,const char *format,
const struct tm *timptr);
```

说明

strftime()把由 format 指向的串所控制的字符放入 s 指向的数组中。format 串包含零个或多个转换说明和普通字符。一个转换说明包括一个%字符和一个确定转换说明动作的转换字符。所有普通字符(包括结尾空字节)不被改变地拷贝到数组中。如果拷贝目标重叠，结果没被定义。多于 maxsize 的字节将不放入数组中。如下所示，每个转换说明被恰当的字符序列替换。恰当的字符序列由程序的本地环境和 timptr 所指向的结构中的值确定。

注：此处只说明中文处理需要增加的部分。

%C 被十进制的世纪号[00～99](年号除以 100 并且取整)替换。

%D 与%m/%d/%y 一样。

%e 被十进制的月中日期[1～31]替换；一个单一数字前加一个空格。

%h 与%b 一样。

%n 被一个换行符替换。

%r 被上午、下午的标记替换；在 POSIX 本地环境中等价于%I:%M:%S%P。

%R 被 24 小时的标记(%H:%M)替换。

%t 被一个制表符替换。

%T 被时间(%H:%M:%S)替换。

可变的转换说明符：

一些转换说明符可以被修改字符 E 或 O 改变，用以说明一个替换格式或说明，而不是用常规的不变的转换说明符描述。如果本地环境的替换格式或说明不存在，那么就象是使用不变的转换说明。

%Ec 被本地环境恰当的日期和时间描述替换。

%EC 被本地环境描述中的基本年的名字替换。

%Ex 被本地环境日期描述替换。

%EX 被本地环境时间描述替换。

%Ey 被相对本地环境描述的%EC(只是年)的偏移量替换。

%EY 被完整年号的描述替换。

%Od 使用本地环境的替换数字符号替换月内日期,如果有零的替换符号,按需要填入前导零,否则填入前导空格。

%Oe 使用本地环境的替换数字符号替换月内日期,按需要填入前导空格。

%OH 使用本地环境的替换数字符号替换小时(24 小时)。

%OI 使用本地环境的替换数字符号替换小时(12 小时)。

%Om 使用本地环境的替换数字符号替换月。

%OM 使用本地环境的替换数字符号替换分钟。

%OS 使用本地环境的替换数字符号替换秒钟。

%Ou 被本地环境描述的表示星期几的数字(星期一=1)替换。

%OU 使用本地环境的替换数字符号替换一年中的星期数(星期天作为一周的第一天,规则与%U 一致)。

%OV 使用本地环境的替换数字符号替换一年中的星期数(星期一作为一周的第一天,规则与%V 一致)。

%Ow 使用本地环境的替换数字符号替换星期的编号(星期天=0)。

%OW 使用本地环境的替换数字符号替换年的星期数(星期一是一周第一天)。

%Oy 使用本地环境的替换数字符号替换本地环境描述的年(相对%C 的偏移值)。

返回

如果结果的字节总数,包括结尾的空字节不超过 maxsize,strfmon()返回 s 指向的数组中放入的字节数,不包括结尾的空字节。否则,返回零,数组的内容不确定。

b) **名称**

strptime —— 日期和时间转换。

格式

```
#include <time.h>
char *strptime (const char *buf,const char *format,struct tm *tm);
```

说明

strptime()利用 format 指定的格式,把 buf 所指的字符串转换成存储于 tm 所指的 tm 结构中的值。

format 由零或多个指令组成。每个指令由下列之一组成:一个或多个空白字符(由 isspace()确定);一个普通字符(不是%,也不是空白字符);或者一个转换说明。每个转换说明由一个%字符后跟指定替换的转换字符组成。两个转换说明之间必须有空白字符或其他非字母数字字符。

返回

如果成功,strptime()返回一指针,该指针指向未经语法分析的第一个字符;否则返回空指针。

c) **名称**

wcsftime —— 把日期和时间转换成宽字符串。

格式

```
#include <wchar.h>
```

size _ t wcsftime (wchar _ t * wcs,size _ t maxsize,const char * format,const struct tm * timptr);

说明

在 format 所指定的串控制下,wcsftime()把宽字符代码存入 wcs 所指的数组中。

该函数看上去象先由 strftime()产生的字符串作为 mbstowcs()的参数,然后 mbstowcs 把结果写入 wcsftime()的宽字符串参数中,最多允许 maxsize 个宽字符代码。

返回

如果结果的宽字符代码数(包括结尾的空宽字符)不大于 maxsize,wcsftime()返回写入 wcs 数组的宽字符代码数(不包括结尾的空宽字符)。否则返回 0,并且数组的内容不确定。如果不成功,置 errno 以示出错。

4.2.7 代码转换

a) **名称**

iconv——代码转换函数。

格式

#include <iconv.h>

size _ t iconv (iconv _ t cd,const char * * inbuf,size _ t * inbytesleft,
char * * outbuf,size _ t * outbytesleft);

说明

iconv()把 inbuf 中一个代码集的一系列字符转换成另一代码集中的一系列对应字符,并写入 outbuf 指向的数组。代码集是由 iconv _ open()调用返回的转换说明符 cd 指定的。参数 inbuf 指向一个变量,该变量指向输入缓冲区中的第一个字符,inbytesleft 描述到缓冲区末尾要转换的字节数。参数 outbuf 指向一变量,该变量指向输出缓冲区中的第一个字节,outbytesleft 描述到缓冲区末尾的有效字节数。

返回

iconv()修改参数所指的变量以反映转换范围,并且返回执行不一致的转换数量。如果输入缓冲区的串完全被转换了,inbytesleft 所指向的值将是零。如果由于输入转换停止了,inbystesleft 将是非零,并且置 errno 以说明情况。如果发生错误,iconv()返回(size _ t)—1,并置 errno 以示出错。

b) **名称**

iconv _ close——代码转换释放函数。

格式

#include <iconv.h>

int iconv _ close(iconv _ t cd);

说明

iconv _ close()释放转换描述符 cd 和其他所有由 inconv _ open()分配的相关资源。如果一个文件描述符被用来实现类型 iconv _ t,该文件描述符将被关闭。

返回

如果成功,返回零值;否则返回—1,并置 errno 以示出错。

c) **名称**

iconv _ open——代码转换分配函数。

格式

#include <iconv.h>

iconv _ t iconv _ open (const char * tocode,const char * fromcode);

说明

iconv_open()返回一个转换描述符，它描述从参数 fromcode 指定的代码集到参数 tocode 所指定的代码集的转换。

在一个进程中，一个转换描述符一直保持有效，直到关闭它为止。如果一个文件描述符被用作转换描述符，FD_CLOEXEC 标志被设置，见<fcntl.h>。

返回

如果成功，iconv_open()返回一个转换描述符以便后续的 iconv()使用；否则返回(iconv_t)-1，并且置 errno 以示出错。

4.2.8 文字对比

a) **名称**

strcoll——用理序信息进行串比较。

格式

```
#include <string.h>
int strcoll (const char *s1,const char *sb);
```

说明

strcoll()将 s1 指向的串与 s2 指向的串进行比较，这种比较基于当前本地环境的 LC_COLLATE。

返回

按当前本地环境的描述解释 s1 和 s2，如果成功，strcoll()根据 s1 所指的串大于、等于或小于 s2 所指的串，返回一个大于、等于或小于零的值；如果发生错误，置 errno，无返回值。

b) **名称**

strxfrm——串转换。

格式

```
#include <string.h>
size_t strxfrm (char * s1,const char *s2,size_t n);
```

说明

strxfrm()转换串 s2，并把结果放入 s1 指向的数组。

转换后使得：如果有两个原始串和两个用 strxfrm()转换后的串，用 strcmp()比较两个转换后的串，它返回的结果与 strcoll()用于同样的两个原始串的结果相同。包括结尾的空符，不多于 n 个的字节放入 s1 指向的数组。如果 n 是零，s1 被允许是空指针。如果转换目标重叠，结果没定义。

返回

如果成功，strxfrm()返回已转换的串的长度(不包括结尾的空符)。如果其值是 n 或更大，s1 指向的数组的内容不确定。如果出现错误，strxfrm()将置 errno 并且无返回值。

c) **名称**

wcscoll——利用理序信息进行宽字符串比较。

格式

```
#include <wchar.h>
int wcscoll (const wchar_t,* ws1,const wchar_t * ws2);
```

说明

wcscoll()将 ws1 指向的宽字符串与 ws2 指向的宽字符串进行比较，这种比较基于当前本地环境的 LC _ COLLATE。

返回

按当前本地环境的描述解释 ws1 和 ws2，如果成功，wcscoll()根据 ws1 所指的宽字符串大于、等于或小于 ws2 所指的宽字符串，返回一个大于、等于或小于零的值；如果发生错误，置 errno，无返回值。

d) **名称**

wcsxfrm——宽字符串转换。

格式

```
#include<wchar.h>
size_t wcsxfrm(wchar_t *ws1,const wchar_t *ws2,size_t n);
```

说明

wcsxfrm()转换宽字符串 ws2，并把结果放入 ws1 指向的数组。

转换后使得：如果有两个原始串和两个用 strxfrm()转换后的串，wcscmp()用于两个转换后的宽字符串，它返回的结果与 wcscoll()用于同样的两个原始宽字符串的结果相同。包括结尾的空宽字符，不多于 n 个的宽字符放入 ws1 指向的数组。

如果 n 是零，ws1 被允许是空指针。如果转换目标重叠，结果没定义。

返回

如果成功，wcsxfrm()返回已转换的宽字符串的长度(不包括结尾的空宽字符)。如果其值是 n 或更大，ws1 指向的数组的内容不确定；如果出现错误，wcsxfrm()将置 errno 并且无返回值。

4.2.9 数字转换

a) **名称**

strtod——把字符串转换成双精度数。

格式

```
#include<stdlib.h>
double strtod(const char *str,char **endptr);
```

说明

strtod()将 str 所指向的串转换成双精度型值。首先它把输入串分成三部分：起始部分，空白字符序列，可以是空；可被解释成浮点常数的主体序列；最后是一个或多个不可识别字符组成的序列，包括输入串结尾的空字节。然后它试图把主体序列转换成浮点数，并返回结果。

主体序列的期望格式是一个可选的＋或－符号，然后是一个可包括基数字符的非空数字序列，最后是一个可选的指数部分。指数部分由 e 或者 E、后随一个可选符号组成，再后是一个或多个十进制数字。主体序列被定义为输入串中最长的起始子序列，从第一个非空白符开始。如果输入串是空或者完全由空白符组成，或者第一个非空白符不是符号、数字或基数，那么主体序列是空。

如果主体序列有期望格式，以第一个数字或基数字符开始的序列被解释为浮点常数象

C 语言中定义的那样，除非用基数字符代替一个小数点(.)，或者既没有指数部分也没有出现基数字符，那么基数字符假定跟在串中的最后一个数字后。如果主体序列以一个减号开始，转换结果是负值。只要 endptr 不是空指针，指向最后的串的指针存储在 endptr 指向的目标中。

基数字符在程序本地环境中定义(LC _ NUMERIC)。在 POSIX 本地环境或者在一个没有定义基数的本地环境中，基数默认为一个小数点(.)。

在不是 POSIX 本地环境中，其他依赖实现的主体序列格式可被接受。

如果主体序列是空或者没有期望格式，不发生转换；只要 endptr 不是空指针，str 的值存储在 endptr 指向的目标中。

返回

如果成功，strtod()返回转换后的值。
如果不发生转换，返回零，并且 errno 被置入[EINVAL]。
如果正确的值超出可表示的范围，返回+/—HUGE _ VAL(根据值的符号)，errno 置入[ERANGE]。
如果正确的值引起下溢，返回零，并将 errno 置入[ERANGE]。

b) **名称**

strtol——把字符串转换成长整数。

格式

```
#include<stdlib.h>
long int strtol(const char *str,char **endptr,int base);
```

说明

strtol()将 str 所指向的串转换成长整型的值。首先它把输入串分成三部分：起始部分，空白字符序列，可以是空；主体序列被解释为以 base 值为基数的整数；最后是一个或多个不可认字符组成的序列，包括输入串结尾的空字节。然后它试图把主体序列转换成长整数，并返回结果。

如果 base 的值是零，主体序列的期望格式是一个十进制常数、八进制常数、或者是十六进制常数，常数前面可带+或—符号。一个十进制常数以非零数字开始，并包含十进制的数字序列。一个八进制常数包含一个前置 0 和随后的 0 到 7 的数字序列。一个十六进制常数包含前置的 0x 或 0X 和随后的十进制数字和代表 10 到 15 的字母 a(A)到 f(F)的序列。

如果 base 的值在 2 到 36 之间，主体序列的期望格式是一字母和数字序列，该序列描述一以 base 值为基数的整数，前面可有+或—符号。字母 a(A)到 z(Z)代表值 10 到 35；只有代表的值小于 base 的字母是允许使用的。如果 base 的值是 16，字母和数字的前面可出现 0x 或 0X，如果有正负号，则在最前面。

主体序列被定义为输入串中最长的起始子序列，从第一个非空白符开始。如果输入字符串是空或者完全由空白符组成，或者第一个非空白符不是符号也不是允许的字母或数字，那么主体序列是空。

如果主体序列具有期望格式而且 base 的值是零，以第一个数字开始的序列被说明为整型常数。如果主体序列具有期望格式而且 base 的值在 2 与 36 之间，则按照 base 的值转换主体序列，每个字母的值如上面给出。如果主体序列以一个减号开始，转换结果是负的。只要 endptr 不是空指针，指向最后的串的指针存储在 endptr 指向的目标中。

在非 POSIX 本地环境中，其他依赖实现的主体序列格式可被接受。如果主体序列是空或者没有期望格式，不发生转换；只要 endptr 不是空指针，str 的值存储在 endptr 指向的目标中。

返回

如果成功，strtol()返回转换后的值。
如果没有转换执行，返回零，并且 errno 被置入[EINVAL]。
如果正确的值溢出，返回 LONG _ MAX 或 LONG _ MIN(根据值的符号)，errno 置入[ERANGE]。

c) **名称**

strtoul——把字符串转换成无符号长整数。

格式

```
#include<stdlib.h>
unsigned long strtoul(const char *str,char **endptr,int base);
```

说明

strtoul()将 str 所指向的串转换成无符号长整型的值。首先它把输入串分成三部分：起始部分，空白字符序列，可以是空；主体序列被解释为以 base 值为基数的整数；最后是一个或多个不可认字符组成的序列，包括输入串结尾的空字节。然后它试图把主体序列转换成无符号长整数，并返回结果。

如果 base 的值是零，主体序列的期望格式是一个十进制常数、八进制常数、或者是十六进制常数，常数前面可带＋或－符号。一个十进制常数以非零数字开始，并包含十进制的数字序列。一个八进制常数包含一个前置 0 和随后的 0 到 7 的数字序列。一个十六进制常数包含前置的 0x 或 0X 和随后的十进制数字和代表 10 到 15 的字母 a(A)到 f(F)的序列。

如果 base 的值在 2 到 36 之间，主体序列的期望格式是一字母和数字序列，该序列描述一以 base 值为基数的整数，前面可有＋或－符号。字母 a(A)到 z(Z)代表值 10 到 35；只有代表的值小于 base 的字母是允许使用的。如果 base 的值是 16，字母和数字的前面可出现 0x 或 0X，如果有正负号，则在最前面。

主体序列被定义为输入串中最长的起始子序列，从第一个非空白符开始。如果输入字符串是空或者完全由空白符组成，或者第一个非空白符不是符号也不是允许的字母或数字，那么主体序列是空。

如果主体序列具有期望格式而且 base 的值是零，以第一个数字开始的序列被说明为整型常数。如果主体序列具有期望格式而且 base 的值在 2 与 36 之间，则按照 base 的值转换主体序列，每个字母的值如上面给出。如果主体序列以一个减号开始，转换结果是负

的。只要 endptr 不是空指针，指向最后的串的指针存储在 endptr 指向的目标中。

在不是 POSIX 本地环境中，其他依赖实现的主体序列格式可被接受。如果主体序列是空或者没有期望格式，不发生转换；只要 endptr 不是空指针，str 的值存储在 endptr 指向的目标中。

返回

如果成功，strtoul()返回转换后的值。
如果没有转换执行，返回零，并且 errno 被置入[EINVAL]。
如果正确的值溢出，返回 LONG _ MAX 或 LONG _ MIN(根据值的符号)，errno 置入[ERANGE]。

d) **名称**

wcstod——把宽字符串转换成双精度数。

格式

```
#include<wchar.h>
double wcstod(const wchar_t *nptr,wchar_t **endptr);
```

说明

wcstod()将 nptr 所指向的宽字符串转换成双精度型的值。首先它把输入宽字符串分成三部分：起始部分，空白宽字符序列，可以是空；可被解释成浮点常数的主体序列；最后是一个或多个不可认宽字符组成的序列，包括输入串结尾的空宽字符。然后它试图把主体序列转换成浮点数，并返回结果。

主体序列的期望格式是一个可选的＋或－符号，然后是一个可包括基数字符的非空数字序列，最后是一个可选的指数部分，指数部分包括 e 或者 E、后随一个可选符号，再后是一个或多个十进制数字。主体序列被定义为输入的宽字符串中最长的起始子序列，从第一个非空白宽字符开始。如果输入的宽字符串是空或者完全由空白宽字符组成，或者第一个非空白符的宽字符不是符号、数字或基数，那么主体序列是空。

如果主体序列有期望格式，以第一个数字或基数字符开始的宽字符序列被解释为浮点常数象 C 语言中定义的那样，除非用基数字符代替一个小数点(.)，或者既没有指数部分也没有出现基数字符，那么基数字符假定跟在宽字符串中的最后一个数字后。如果主体序列以一个减号开始，转换结果是负值。只要 endptr 不是空指针，指向最后的宽字符串的指针存储在 endptr 指向的目标中。

基数字符在程序本地环境中定义(LC _ NUMERIC)。在 POSIX 本地环境或者在一个没有定义基数的本地环境中，基数默认为一个小数点(.)。

在不是 POSIX 本地环境中，其他依赖实现的主体序列格式可被接受。

如果主体序列是空或者没有期望格式，不发生转换；只要 endptr 不是空指针，nptr 的值存储在 endptr 指向的目标中。

返回

wcstod()返回转换后的值。如果不发生转换，返回零，并且 errno 被置入[EINVAL]。

如果正确的值超出可表示的范围，返回+/—HUGE_VAL(根据值的符号)，errno 置入[ERANGE]。如果正确的值引起下溢，返回零，并将 errno 置入[ERANGE]。

e) **名称**

wcstol——把宽字符串转换成长整数。

格式

```
#include<wchar.h>
long int wcstol(const wchar_t *nptr,wchar_t **endptr,int base);
```

说明

wcstol()将 nptr 所指向的宽字符串转换成长整型的值。首先它把输入的宽字符串分成三部分：起始部分，空白字符序列，可以是空；主体序列被解释为以 base 值为基数的整数；最后是一个或多个不可认字符组成的序列，包括输入宽字符串结尾的空宽字符。然后它试图把主体序列转换成长整数，并返回结果。

如果 base 的值是零，主体序列的期望格式是一个十进制常数、八进制常数、或者是十六进制常数，常数前面可带+或—符号。一个十进制常数以非零数字开始，并包含十进制的数字序列。一个八进制常数包含一个前置 0 和随后的 0 到 7 的数字序列。一个十六进制常数包含前置的 0x 或 0X 和随后的十进制数字和代表 10 到 15 的字母 a(A)到 f(F)的序列。

如果 base 的值在 2 到 36 之间，主体序列的期望格式是一字母和数字序列，该序列描述一以 base 值为基数的整数，前面可有+或—符号。字母 a(A)到 z(Z)代表值 10 到 35；只有代表的值小于 base 的字母是允许使用的。如果 base 的值是 16，字母和数字的前面可出现 0x 或 0X，如果有正负号，则在最前面。

主体序列被定义为输入宽字符串中最长的起始子序列，从第一个非空白宽字符开始。如果输入宽字符串是空或者完全由空白宽字符组成，或者第一个非空白宽字符不是符号也不是允许的字母或数字，那么主体序列是空。

如果主体序列具有期望格式而且 base 的值是零，以第一个数字开始的宽字符序列被说明为整型常数。如果主体序列具有期望格式而且 base 的值在 2 与 36 之间，则按照 base 的值转换主体序列，每个字母的值如上面给出。如果主体序列以一个减号开始，转换结果是负的。只要 endptr 不是空指针，指向最后的宽字符串的指针存储在 endptr 指向的目标中。

在不是 POSIX 本地环境中，其他依赖实现的主体序列格式可被接受。

如果主体序列是空或者没有期望格式，不发生转换；只要 endptr 不是空指针，nptr 的值存储在 endptr 指向的目标中。

返回

如果成功，wcstol()返回转换后的值。如果没有转换执行，返回零，并且 errno 被置入[EINVAL]。如果正确的值溢出，返回 LONG_MAX 或 LONG_MIN(根据值的符号)，errno 置入[ERANGE]。

f) **名称**

wcstoul——把宽字符串转换成无符号长整数。

格式

#include<wchar.h>

unsigend long int wcstoul (const wchar_t *nptr,wchar_t **endptr,int base);

说明

wcstoul()将 nptr 所指向的宽字符串转换成无符号长整型的值。首先它把输入宽字符串分成三部分:起始部分,空白字符序列,可以是空;主体序列被解释为以 base 值为基数的整数;最后是一个或多个不可认字符组成的序列,包括输入宽字符串结尾的空宽字符。然后它试图把主体序列转换成无符号长整数,并返回结果。

如果 base 的值是零,主体序列的期望格式是一个十进制常数、八进制常数、或者是十六进制常数,常数前面可带+或-符号。一个十进制常数以非零数字开始,并包含十进制的数字序列。一个八进制常数包含一个前置 0 和随后的 0 到 7 的数字序列。一个十六进制常数包含前置的 0x 或 0X 和随后的十进制数字和代表 10 到 15 的字母 a(A)到 f(F)的序列。

如果 base 的值在 2 到 36 之间,主体序列的期望格式是一字母和数字序列,该序列描述一以 base 值为基数的整数,前面可有+或-符号。字母 a(A)到 z(Z)代表值 10 到 35;只有代表的值小于 base 的字母是允许使用的。如果 base 的值是 16,字母和数字的前面可出现 0x 或 0X,如果有正负号,则在最前面。

主体序列被定义为输入宽字符串中最长的起始子序列,从第一个非空白宽字符开始。如果输入的宽字符串是空或者完全由空白宽字符组成,或者第一个非空白宽字符不是符号也不是允许的字母或数字,那么主体序列是空。

如果主体序列具有期望格式而且 base 的值是零,以第一个数字开始的序列被说明为整型常数。如果主体序列具有期望格式而且 base 的值在 2 与 36 之间,则按照 base 的值转换主体序列,每个字母的值如上面给出。如果主体序列以一个减号开始,转换结果是负的。只要 endptr 不是空指针,指向最后的宽字符串的指针存储在 endptr 指向的目标中。

在不是 POSIX 本地环境中,其他依赖实现的主体序列格式可被接受。

如果主体序列是空或者没有期望格式,不发生转换;只要 endptr 不是空指针,nptr 的值存储在 endptr 指向的目标中。

返回

如果成功,wcstoul()返回转换后的值。

如果没有转换执行,返回零,并且 errno 被置入[EINVAL]。

如果正确的值溢出,返回 LONG_MAX 或 LONG_MIN(根据值的符号),errno 置入[ERANGE]。

4.2.10 扫描功能

a) **名称**

fscanf,scanf,sscanf——转换格式化输入。

格式

```
#include<stdio.h>
int fscanf (FILE *stream,const char *format,...);
int scanf (const char *format,...);
int sscanf (const char *s,const char *format,...);
```

说明

注：此处只说明有关中文处理的部分。

fscanf()从命名的输入流中读；scanf()从标准输入流 stdin 中读；sscanf()从串 s 中读。每个函数按照格式解释读入的字节，并将结果存储到它的参数中。每个函数都要求一些参数，其中包括下文描述的控制串 format 以及一组指针型参数，它指示转换的输入的存储位置。如果没有足够的参数用于格式转换，结果没定义。如果参数超过格式转换所需要的数目，多余的参数被求值或被忽略。

%n$将代替转换说明符%，n 是[1,(NL_ARGMAX)]中的一个十进制整数，它给出参数序列中参数的位置。转换说明符适用于 format 后面参数序列中的 n 个参数，而不适用于再下面没有用到的参数。这种特点为特定语言按次序选择参数提供了格式定义。在含有%n$格式转换说明的格式串中，对于参数表中已编号的参数是否可以多次从参数表中指定未做规定。

format 可以含有转换说明%或%n$，但通常两种格式不能混用于一个 format 串中。只有%%或%*例外可以与%n$格式混用。

fscanf()的所有格式都允许对输入串中依赖语言的基数字符的检测，基数字符是在程序的本地环境里定义的(LC_NUMERIC)，在 POSIX 本地环境或在没有定义基数字符的本地环境中，基数字符默认为小数点(.)。

下列是有效的转换字符(只列出与中文处理有关的转换字符)：

C　匹配由域宽指定其字符数的字符序列(如果指令中未指定域宽，则为 1)。该序列以与 mbstowcs()同样的方式被转换成宽字符代码。对应的参数应该是指向 wchar_t 型数组的开始宽字符代码的指针，数组必须足够大，以便能接受该序列，结尾不加空宽字符。如果匹配序列以初始切换状态开始，转换类似于函数 mbstowcs()；否则转换无定义。

S　匹配一个不是空白符的字符序列。该序列以与 mbstowcs()同样的方式被转换成宽字符代码。对应的参数应该是指向 wchar_t 型数组的开始宽字符代码的指针，数组必须足够大，以便能接受该序列和结尾的空宽字符，该空宽字符将被自动添加。如果域宽已经指定，它指示接收的字符的最大数。

返回

如果成功，这些函数返回被成功匹配并赋值的输入项的数目；如果匹配很早失败，返回值可能是零。如果在第一次匹配失败或转换之前输入结束，返回 EOF。

如果发生读错误，置流的出错信息指示器，返回 EOF，置 errno 以示出错。

4.2.11　区域环境设定

a) **名称**

setlocale()——设置程序的本地环境。

格式

```
#include<locale.h>
char *setlocale (int category,const char *locale);
```

说明

setlocale 根据 category 和 locale 选择适当的程序本地环境，可被用来改变或查询程序本地环境的全部或部分。category 的 LC_ALL 值表示程序的全部本地环境；category 的其他值表示程序的部分本地环境。

LC_COLLATE	影响正则表达式和理序函数的行为。
LC_CTYPE	影响正则表达式、字符分类、字符转换函数、宽字符函数的行为。
LC_MESSAGES	影响命令和实用程序要求的作为肯定或否定回答的串的形式，以及消息的内容。
LC_MONETARY	影响处理货币值的函数的行为。
LC_NUMERIC	影响格式化输入/输出函数的基数字符和串转换函数。
LC_TIME	影响时间转换函数的行为。

locale 参数是一指向字符串的指针，包含 category 所需的设置，该串的内容是依赖于实现的。下列 locale 的预置值是为所有 category 的设置定义的。

"POSIX"	为 C 语言翻译指定的最小环境称为 POSIX 本地环境。如果 setlocale()没被调用，POSIX 即为默认值。
"C"	同"POSIX"。
""	指定一种依赖于实现的本地环境。它与相关的环境变量 LC_* 和 LANG 的值相一致。
空指针	直接用 setlocale 去询问当前国际化环境并返回 locale 的名字。

返回

如果成功，setlocale()返回与新本地环境的类型相关的串。否则返回空指针，并且程序本地环境没被改变。

如果 locale 是空指针，则 setlocale()返回程序当前本地环境中与 category 相关的串。而程序的本地环境不变。

后续的调用可以利用 setlocale()返回的串和与它有关的 category 来恢复程序本地环境的相应部分。返回串不能被程序修改，但可以被后面的 setlocale()调用盖写。

4.2.12 宽字符分类

a) **名称**

iswalpha——检测一个宽字符的字母。

格式

```
#include<wchar.h>
int iswalpha(wint_t wc);
```

说明

iswalpha()检测 wc 是否是在程序当前本地环境中定义的 alpha 类宽字符代码。

任何情况下，wc 是 wint_t 类型的，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是字母的宽字符代码，返回非零；否则返回零。

b) **名称**

iswupper——检测一个宽字符的大写字母。

格式

```
#include<wchar.h>
int iswupper(wint_t wc);
```

说明

iswulpper()检测 wc 是否是在程序当前本地环境中定义的 upper 类宽字符代码。

任何情况下,wc 是 wint_t 类型的,它的值必须是与当前本地环境有效字符相对应的宽字符代码,或者等于宏 WEOF。如果是其他的值,没有定义。

返回

如果 wc 是大写字母的宽字符代码,返回非零;否则返回零。

c) **名称**

iswlower——检测一个宽字符的小写字母。

格式

```
#include<wchar.h>
int iswiower(wint_t wc);
```

说明

iswlower()检测 wc 是否是在程序当前本地环境中定义的 lower 类宽字符代码。
任何情况下,wc 是 wint_t 类型的,它的值必须是与当前本地环境有效字符相对应的宽字符代码,或者等于宏 WEOF。如果是其他的值,没有定义。

返回

如果 wc 是小写字母的宽字符代码,返回非零;否则返回零。

d) **名称**

iswdigit——检测一个宽字符的十进制数。

格式

```
#include<wchar.h>
int iswdigit(wint_t wc);
```

说明

iswdigit()检测 wc 是否是在程序当前本地环境中定义的 digit 类宽字符代码。

任何情况下,wc 是 wint_t 类型的,它的值必须是与当前本地环境有效字符相对应的宽字符代码,或者等于宏 WEOF。如果是其他的值,没有定义。

返回

如果 wc 是小写字母的宽字符代码,返回非零;否则返回零。

e) **名称**

iswxdigit——检测一个宽字符的十六进制数。

格式

```
#include<wchar.h>
int iswxdigit(wint_t wc);
```

说明

iswxdigit()检测 wc 是否是在程序当前本地环境中定义的 xdigit 类宽字符代码。

任何情况下，wc 是 wint_t 类型的，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个十六进制数宽字符代码，返回非零；否则返回零。

f) **名称**

iswalnum——检测宽字符的一个字母数字。

格式

```
#include<wchar.h>
int iswalnum(wint_t wc);
```

说明

iswalnum()检测 wc 是否是在程序当前本地环境中定义的 alpha 或者 digit 类宽字符代码。

任何情况下，wc 是 wint_t 类型的，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个十字母数字宽字符代码，返回非零；否则返回零。

g) **名称**

iswspace——检测一个空白宽字符。

格式

```
#include<wchar.h>
int iswspace(wint_t wc);
```

说明

iswspace()检测 wc 是否是在程序当前本地环境中定义的 space 类宽字符代码。

任何情况下，wc 是 wint_t 类型的，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个空白宽字符代码，返回非零；否则返回零。

h) **名称**

iswpunct——检测宽字符的一个标点符号。

格式

```
#include<wchar.h>
int iswpunct(wint_t wc);
```

说明

iswpunct()检测 wc 是否是在程序当前本地环境中定义的 punct 类宽字符代码。

任何情况下，wc 是 wint_t 类型，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个标点符号的宽字符代码，返回非零；否则返回零。

i) **名称**

iswprint——检测一个可打印的宽字符。

格式

```
#include<wchar.h>
int iswprint(wint_t wc);
```

说明

iswprint()检测 wc 是否是在程序当前本地环境中定义的 print 类宽字符代码。

任何情况下，wc 是 wint_t 类型，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个可打印的宽字符代码，返回非零；否则返回零。

j) **名称**

iswgraph——检测一个可显示的宽字符。

格式

```
#include<wchar.h>
int iswgraph(wint_t wc);
```

说明

iswgraph()检测 wc 是否是在程序当前本地环境中定义的 graph 类宽字符代码。

任何情况下，wc 是 wint_t 类型，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个可显示的宽字符代码，返回非零；否则返回零。

k) **名称**

iswcntrl——检测宽字符的一个控制符。

格式

```
#include<wchar.h>
int iswcntrl(wint_t wc);
```

说明

iswcntrl()检测 wc 是否是在程序当前本地环境中一个表示 control 类的宽字符代码。

任何情况下，wc 是 wint_t 类型的，它的值必须是与当前本地环境有效字符相对应的宽字符代码，或者等于宏 WEOF。如果是其他的值，没有定义。

返回

如果 wc 是一个控制宽字符代码，返回非零；否则返回零。

l) **名称**

wctype——定义字符类别。

格式

```
#include<wchar.h>
wctype_t wctype(const char *charclass);
```

说明

wctype()是为在当前本地环境中定义有效字符分类名而定义的。charclass 是一个标识普通字符分类的串，它需要特定代码集类型信息。下列字符分类名在所有本地环境中被定义——"alnum"，"alpha"，"blank"，"cntrl"，"digit"，"graph"，"lower"，"print"，"

punct","space","upper"和"xdigit"。

在本地环境的定义文件(LC_CTYPE)中也可以指定附加的字符分类名。

函数返回 wctype_t 型的值,它可以被用作第二个参数去调用 iswctype()。根据程序本地环境中字符类型信息所定义的编码字符集规则,wctype()确定 wctype_t 型的值。wctype()的返回值是有效的,直到调用 setlocale()修改了 LC_CTYPE 为止。

返回

如果为当前本地环境给出的字符分类名是无效的,则 wctype()返回零;否则返回一个可被 iswctype()使用的 wctype_t 类型的值。

m) **名称**

iswctype——检测指定分类字符。

格式

```
#include<wchar.h>
int iswctype_t(wint_t wc,wctype_t charclass);
```

说明

iswctype()确定宽字符代码 wc 是否属于字符类 charclass,返回真或假。wc 可以是 WEOF 和与当前本地环境中有效字符编码相对应的宽字符代码。如果参数 wc 不在函数的定义域内,结果没定义。如果 charclass 的值是无效的(即无法通过调用 wctype()获得,或者调用 setlocale()影响了 LC_CTYPE 而使 charclass 无效),结果是依赖于实现的。

返回

如果失败,iswctype()返回零;否则返回非零。

4.2.13 文字大小写转换

a) **名称**

towupper()——把小写的宽字符代码转换成大写。

格式

```
#include<wchar.h>
wint_t towupper(wint_t wc);
```

说明

towupper()有个 wint_t 类型的定义域,它的值必须是可表示为 wchar_t 类型的字符,而且必须是与当前本地环境中有效字符相对应的宽字符代码或是 WEOF 的值。如果是其他的值,行为没被定义。如果 towupper()的参数表示一小写宽字符代码,并且存在相应的大写宽字符代码(被程序本地环境 LC_CTYPE 的字符类型信息定义的),执行结果是相应的大写宽字符代码。所有定义域内的其他参数值无改变返回。

返回

如果成功,towupper()返回与参数相对应的大写字母;否则返回没做改变的参数。

b) **名称**

towlower()——把大写的宽字符代码转换成小写。

格式

```
#include<wchar.h>
wint_t towlower(wint_t wc);
```

说明

towlower()有个 wint_t 类型的定义域,它的值必须是可表示为 wchar_t 类型的字符,

而且必须是与当前本地环境中有效字符相对应的宽字符代码或是 WEOF 的值。如果是其他的值，行为没被定义。如果 towlower()的参数表示一大写宽字符代码，并且存在相应的大写宽字符代码(被程序本地环境 LC _CTYPE 的字符类型信息定义的)，执行结果是相应的小写宽字符代码。所有定义域内的其他参数值无改变返回。

返回

如果成功，towlower()返回与参数相对应的小写字母；否则返回没做改变的参数。

4.2.14 宽字符宽度

a) **名称**

wcwidth()——计算一宽字符代码的宽度。

格式

```
#include<wchar.h>
int wcwidth(wint_t wc);
```

说明

wcwidth()确定宽字符 wc 所占列位置的数目。wc 的值必须是一个可表示为 wchar_t 类型的字符，而且必须是一个与当前本地环境中有效字符相对应的宽字符代码。

返回

wcwidth()或者返回零(如果 wc 是空宽字符)，或者返回宽字符代码 wc 的宽度，或者返回－1(如果 wc 不对应一个可打印字符代码)。

b) **名称**

wcswidth()——计算一宽字符串的宽度。

格式

```
#include<wchar.h>
int wcswidth(const wint_t * pwcs,size_t n);
```

说明

wcswidth()确定由 pwcs 指向的串的 n 个宽字符代码所占列位置的数目(如果在 n 个宽字符代码之前遇到空宽字符，就少于 n 个宽字符代码)。

返回

wcswidth()返回零(如果 pwcs 指向一个空宽字符)，或者返回 pwcs 所指向的宽字符串的宽度，或者返回－1(如果 pwcs 所指向的宽字符串中的任何宽字符不是可打印宽字符)。

4.2.15 出错处理

a) **名称**

strerror——获得出错信息串。

格式

```
#include<string.h>
char * strerror(int errnum);
```

说明

strerror()把 errnum 中的出错号映射成依赖本地环境的出错信息串，并且返回指向串的指针。所指向的串不能由程序修改，但可以被后续的 strerror()或 popen()函数调用盖写

strerror()返回的出错信息串的内容是由当前本地环境的 LC _MESSAGES 设置决定的。

返回

如果成功，strerror()返回一个指向获得的信息串的指针。
如果出错，置 errno，但无返回值。

b) **名称**

catopen——打开一个信息目录。

格式

```
#include<nl_types.h>
nl_catd catopen(const char *name,int oflag);
```

说明

catopen()打开一个信息目录并返回信息目录描述字。name 指定要打开目录的名字。如果 name 中含有"/"，name 指定的是全路径名。否则就使用环境变量 NLSPATH。

如果 NLSPATH 在当前环境中不存在，或者 NLSPATH 所指定的任何部分都找不到信息目录，一个依赖于实现的默认路径将被使用。如果 oflag 的值是 NL_CAT_LOCATE 默认值受 LC_MESAGES 的设置影响，或者如果 oflag 是 0，默认值受环境变量 LANG 影响。

一个信息目录描述字在进程关闭它或者 exec 族中的任一函数的成功调用之前保持有效。改变 LC_MESSAGES 目录的设置使当前打开的目录无效。

如果一个文件描述字用作信息目录描述字，FD_CLOEXEC 标记被设置。如果 oflag 的值是 0，环境变量 LANG 用于定位目录而不考虑 LC_MESSAGES。如果 oflag 是 NL_CAT_LOCATE，LC_MESSAGES 用于定位信息目录。

返回

如果成功，catopen()返回一个信息描述字以便调用 catgets()和 catclose()时使用；否则 catopen()返回－1 并且置 errno 以示出错。

c) **名称**

catclose——关闭一个信息目录描述字。

格式

```
#include<nl_types.h>
int catclose(nl_catd catd);
```

说明

catclose()关闭 catd 指定的信息目录。如果一个文件描述字是 nl_catd 类型的，那么该文件描述字将被关闭。

返回

如果成功，catclose()返回一个零值；否则返回－1，置 errno 以示出错。

d) **名称**

catgets——读取一条程序信息。

格式

```
#include<nl_types.h>
char *catgets(nl_catd catd,int set_id,int msg_id,const char *s);
```

说明

catgets()试图从 catd 标识的信息目录中读集合 set_id 中的信息 msg_id。参数 catd 是

前面调用 catopen()时返回的信息目录描述字。s 指向一个默认的信息串；如果不能检索出标识的信息，则由 catgets()返回该信息串。

返回

如果成功地检索出标识的信息，catgets()返回一个指向内部缓冲区的指针，包括由空符结尾的信息串。如果不成功返回 s。

4.2.16 扩展中文字符类功能

a) **名称**

fctohc——把一个全角字符转换成对应的半角字符。

格式

```
#include<stdlib.h>
int fctohc(int *pc,const char *s,size_t n);
```

说明

如果 s 不是空指针，那么 fctohc()先确定由 s 指向的字符字节数，然后确定与该字符相对应的半角字符代码值(与空字符等价的半角字符代码值是零)。如果该字符是有效的并且 pc 不是空指针，fctohc()把半角字符代码值存放在 pc 指向的目标中。

本函数受当前本地环境 LC_CTYPE 影响。对于依赖于状态的编码，通过将参数 s 置为空来调用本函数，使函数设置为初始状态；后续的调用，如果参数 s 非空，则使函数的初始状态根据需要改变。如果编码依赖于状态，那么参数 s 为空指针的调用使本函数返回一个非零值，否则返回零。改变当前的 LC_CTYPE 使本函数的初始状态不确定。s 指向的数组中至多 n 个字节被检查。

返回

当 s 是空指针时，如果字符编码依赖于状态，那么函数 fctohc()返回一个非零值；如果编码不依赖于状态，则函数返回零值。当 s 是非空指针时，如果 s 指向空字节，那么 fctohc()返回零；或者如果 n 个或少于 n 个字节组成了合法的多字节字符，则返回多字节字符的字节数；或者如果是无效字符，则函数返回−1 并置 errno 以示出错。
不可能返回大于 n 或者宏 MB_CUR_MAX 的值。

b) **名称**

hctofc——把一个半角字符转换成对应的全角字符。

格式

```
#include<stdlib.h>
int hctofc(char *s,int c);
```

说明

hctofc()确定 c 表示的半角字符所对应的全角字符字节数，并把对应的全角字符存放在 s(如果 s 不是空指针)指向的目标数组中。最多存放 MB_CUR_MAX 个字节。如果 c 的值为零，hctofc()被置为初始切换状态。

本函数受当前本地环境 LC_CTYPE 影响。对于依赖于状态的编码，通过将参数 s 置为空来调用本函数，使函数设置为初始状态；后续的调用，如果参数 s 非空，则使函数的初始状态根据需要改变。如果编码依赖于状态，那么参数 s 为空指针的调用使本函数返回一个非零值，否则返回零。改变当前的 LC_CTYPE 使本函数的初始状态不确定。

返回

当 s 是空指针时，如果字符编码依赖于状态，那么函数 hctofc()返回一个非零值；如果

编码不依赖于状态，则函数返回零值。当 s 是非空指针时，如果 c 不对应合法的字符，那么函数返回－1；否则返回 c 对应的字节数。

不可能返回大于宏 MB _ CUR _ MAX 的值。

c) **名称**

isfullc——检测多字节字符的一个全角字符。

格式

```
#include<ctype.h>
int isfullc(char *pc,size_t n);
```

说明

isfullc()检测 pc 指向的字符是否是在程序当前本地环境中定义的 fullc 类宽字符代码。任何情况下，pc 指向的字符值必须是与当前本地环境有效字符相对应的多字节字符代码，或者等于宏 EOF。如果是其他的值，没有定义。最多 n 个字节被检查。

返回

如果 pc 指向的字符是一个全角字符的多字节字符代码，返回非零；否则返回零。

d) **名称**

isfphonogram——检测多字节字符的一个注音字符。

格式

```
#include<ctype.h>
int isfphonogram(char *pc,size_t n);
```

说明

isfphonogram()检测 pc 指向的字符是否是在程序当前本地环境中定义的 fphonogram 类宽字符代码。任何情况下，pc 指向的字符值必须是与当前本地环境有效字符相对应的多字节字符代码，或者等于宏 EOF。如果是其他的值，没有定义。最多 n 个字节被检查。

返回

如果 pc 指向的字符是一个注音字符的多字节字符代码，返回非零；否则返回零。

e) **名称**

isradical——检测多字节字符的一个部首字符。

格式

```
#include<ctype.h>
int isradical(char *pc,size_t n);
```

说明

isradical()检测 pc 指向的字符是否是在程序当前本地环境中定义的 radical 类宽字符代码。任何情况下，pc 指向的字符值必须是与当前本地环境有效字符相对应的多字节字符代码，或者等于宏 EOF。如果是其他的值，没有定义。最多 n 个字节被检查。

返回

如果 pc 指向的字符是一个部首字符的多字节字符代码，返回非零；否则返回零。

f) **名称**

isundefchar——检测多字节字符的一个未定义字字符。

格式

```
#include<ctype.h>
int isundefchar(char *pc,size_t n);
```

说明

isundefchar()检测 pc 指向的字符是否是在程序当前本地环境中定义的 undefchar 类宽字符代码。任何情况下,pc 指向的字符值必须是与当前本地环境有效字符相对应的多字节字符代码,或者等于宏 EOF。如果是其他的值,没有定义。最多 n 个字节被检查。

返回

如果 pc 指向的字符是一个未定义字字符的多字节字符代码,返回非零;否则返回零。

4.3 本地化

所谓本地化(Localization)是向特定本地语言操作环境的转换。本标准只约束中文化。

4.3.1 代码体系

a) 中文化工作,实现者应给出 GB 2312 代码集在 UNIX 系统的代码体系中的正确表示。

b) GB/T 13000.1(UCS)

如果 UNIX 系统采用 UCS 代码体系,那么中文化工作应提供 GB 2312 与 UCS 码制转换手段。

4.3.2 国家特征文件

国家特征文件的编写属于本地化工作。

本标准与《POSIX 中文特征文件》(见附录 A)完全兼容。

4.4 I/O 服务

4.4.1 输入

输入方法子系统可按其功能分为:

a) 输入管理层

根据 Locale 环境自动到指定目录下找出所需的输入方法模块,将其启动或装入应用程序,并填写有关入口表,以便建立应用程序与输入方法之间的联系。

b) 前端处理层

对各个输入单元进行管理,负责解释特殊键的含义,切换输入单元,并把输入单元处理的结果存放在相应的缓冲区中,以便应用程序取走,或调辅助区处理层函数显示。

c) 输入单元层

根据输入单元的要求,对每个输入键事件进行解释,确定相应的动作,形成符合输入单元要求的输入串,并调用输入单元算法层的函数进行字典查找运算或代码转换运算,把查找结果返回前端处理层。

d) 输入单元算法层

根据上层函数送来的输入码串进行字典查找运算,并把查找结果返回上一层函数。

e) 辅助区处理层

提供不同风格的状态区、预编辑区、选字区的处理函数,以便向用户提供不同风格的输入界面。

其中,第 a、b、e 层构成了输入方法的应用程序界面(API)。第 e 层,即辅助区处理层应由应用程序实现,其界面应符合本标准要求。附录 B 提供了一个实现的模型。

4.4.1.1 界面规格

4.4.1.1.1 输入管理层

输入管理层的功能是根据 Locale 环境变量的内容自动到指定目录下查找所需的输入方法模块,并将其启动或装入应用程序,填写有关入口表,以便建立应用程序与输入方法之间的联系。此层函数主要提供给 Input Server、Input Daemon 或应用程序使用。

a) **名称**

imopen()——打开输入方法。

格式

IM _ Des imopen(char * Language);

说明

根据指定本地环境的内容找出相应的输入方法模块，将其启动或装入应用程序，并对该输入方法进行初始化，以便在应用程序和输入方法之间建立联系。

Language 本地环境。默认值是当前环境。

返回

成功则返回输入方法描述字 IM _ Des；否则返回－1。

b) **名称**

imclose()——关闭输入方法。

格式

int imclose(IM _ Des im);

说明

释放输入方法占用的资源，断开与输入方法模块的连接。

im 输入方法描述字。

返回

成功则返回 0；否则返回－1。

4.4.1.1.2 前端处理层

前端处理层对各个输入单元进行管理(如创建、关闭等)，负责解释特殊键的含义，切换输入单元，并把输入单元处理的结果存放在相应的缓冲区中，使应用程序可用，或调用辅助区处理层函数进行显示处理。此层函数提供给 Input Server、Input Daemon 或应用程序开发输入方法模块使用。

a) **名称**

imcreat()——创建输入对象。

格式

IM _ unit imcreat(IM _ Des im,IM _ cb * im _ cb,caddr _ t udata);

说明

为指定的输入方法创建一个输入对象，该输入对象是一个由应用程序和输入方法模块共用的数据结构，其主要包括：预编辑区内容、选字区内容、状态区内容、输入方法的转换结果以及辅助区处理层函数入口地址表等内容。一个输入方法可以有几个输入对象。

im 指定的输入方法描述字。

im _ cb 指向应用程序给出的辅助区处理层函数入口表的指针。

udata 应用程序向辅助区处理层函数传递的数据。输入方法对此数据不进行任何改动，只是将其原样传给辅助区处理层函数，由辅助区处理层函数加以解释，其中可以包含有 font ID、输入界面风格及位置等信息。成功则返回该输入方法公用数据结构 IM _ unit 指针；否则返回 NULL。

b) **名称**

imdestroy()——关闭输入对象。

格式

void imdestroy(IM _ unit imp);

说明

关闭指定的输入对象并释放相应资源。

imp　指向要关闭的输入对象。

返回

无。

c) **名称**

imfilter()——输入对象过滤程序。

格式

```
int imfilter (IM _ unit imp,caddr _ t inputdata,
        unsigned char * string,int * bytesofstring);
```

说明

对输入的数据进行处理。如果输入数据用作输入方法的内部处理，则由 imfilter 实现其相应功能，而不将该数据传入应用程序。所有输入的数据首先要经过 imfilter 处理。这样，本函数起了一个数据过滤的作用。内部处理过程中，通过 imp 中的相应域调用辅助区处理层函数，以进行输入界面管理。

imp　指向输入对象。

inputdata　输入数据，随各输入设备的不同而不同。

string　若转换成汉字则返回其文件码(内码)，否则返回 NULL，表示还没有汉字出现。

返回

若返回 InputUsed，则表示输入数据用作输入方法的内部处理；

若返回 InputNotUsed，则表示输入方法不使用该数据。

d) **名称**

imioctl()——控制、操作输入对象。

格式

```
int imioctl(IM _ unit imp,int operation,caddr _ t arg);
```

说明

根据要求可以对输入对象进行一系列的控制或查询操作。

imp　指向输入对象。

operation　要求的操作。根据中文输入的需要，可以分别对预编辑区进行控制和查询，可支持的操作有：

IM _ LOAD　请求装入一个输入单元，一旦装入后，才可以使用该输入法。要装入的输入单元由 arg 指定，若 arg 为 NULL，则装入默认的输入单元。

IM _ UNLOAD　请求卸出一个输入单元，要卸出的输入单元由 arg 指定，若 arg 为 NULL，则卸出除 ASCII 外所有的输入单元。

IM _ CLEAR　清除预编辑区或选字区。

IM _ RESIZE　改变预编辑区、状态区或选字区的最大长度。

IM _ QUERY　根据 arg 结构指定的查询内容标志，分别查询状态区、预编辑区以及选字区的信息(如输入单元状态、预编辑正文、编辑光标位

置等),并将返回信息存于 arg 结构之中。

arg　　operation 的参数。

返回

返回 0 表示成功;−1 表示失败,并给 IMErrno 赋值以示出错。

4.4.1.1.3　输入单元层

输入单元层的功能是根据前端处理层函数传送来的输入码串进行查找/转换运算,并把查找结果返回前端处理函数。输入单元加入到系统中时,则必须符合本条要求。

下面函数名中的 xx 可用输入单元名替换。

a) **名称**

xxopen()——打开输入单元。

格式

```
im_t xxopen(char *im_name);
```

说明

为该输入单元的使用设置初值,如装入索引、分配空间等。

im_name　输入单元的名称。

返回

成功返回该输入单元描述字;否则返回−1。

b) **名称**

xxclose()——关闭输入单元。

格式

```
int xxclose(im_t im_d);
```

说明

释放该输入单元占用的资源,关闭该输入单元。

im_d　输入单元描述字。

返回

成功则返回 0,否则返回−1。

c) **名称**

xxfilter()——输入单元过滤程序。

格式

```
int xxfilter(im_t im_d, caddr_t inputdata, unsigned char *string, int *bytesofstring);
```

说明

输入单元解释用户输入的内容,并根据输入单元的规定进行必要的转换。

im_d	输入单元描述字。
inputdata	输入的数据。
string	转换后的汉字内码串。
bytesofstring	返回的串长度。

返回

InputUsed	表示该输入单元能处理输入的数据。

InputNotUsed　　表示该输入单元不使用输入的数据。

d) **名称**

xxioctl()——控制、操作输入单元。

格式

int xxioctl(im _ t im _ d,int operation,caddr _ t arg);

说明

根据指定的操作改变输入单元的状态。

im _ d　　输入单元描述字。

operation　　要进行的操作，主要有：

IM _ START　　置 im _ d 所指定的输入单元为正在使用。

IM _ STOP　　置 im _ d 所指定的输入单元为暂停使用。

arg　　operation 的参数。

返回

成功则返回 0，否则返回－1。

4.4.1.1.4　辅助区处理层

辅助区处理层主要提供输入方法界面，即对预编辑区、状态区以及选字区进行相应的管理。输入方法通过本层函数与用户进行交互联系。

根据输入界面的类型，可以将本层函数分成三类：

——预编辑区管理

输入方法使用此类函数在预编辑的过程(即汉字编码输入过程)中显示预编辑区正文，此类函数有：

textdraw　　在预编辑区中显示正文字符串。

textclear　　清除预编辑区中的正文字符串。

——状态区管理

输入方法使用状态区管理函数来进行状态区的操作。此类函数有：

statedraw　显示状态信息。

stateclear　清除状态信息。

——　选字区管理

输入方法使用选字区管理函数与用户进行复杂的对话(比如重码的选择)，

选字区可根据不同应用程序的需要进行不同的设置。此类函数包括：

auxdraw　显示选字区信息。

auxclear　清除选字区信息。

本层函数可由应用程序或输入方法提供，并在创建输入对象时将入口表填入 IM _ unit 数据结构。

a) **名称**

textdraw(　)——显示预编辑区中的正文字符串。

格式

int textdraw(IM _ unit imp,char * text,caddr _ t udata);

说明

在预编辑(即汉字编码输入)过程中，输入方法使用 textdraw 在预编辑区中显示正文字符串。

imp　　指定输入对象。

text　　预编辑正文字符串。

udata　　应用程序信息。

返回

返回 0 表示成功；返回—1 表示失败，并为 IMErrno 全局变量赋值以示出错。

b) **名称**

textclear(　)——清除预编辑区中的正文字符串。

格式

int textclear(IM _ unit imp,caddr _ t udata);

说明

清除指定输入对象的预编辑正文区。

imp　　指定输入对象。

udata　　应用程序信息。

返回

返回 0 表示成功；返回—1 表示失败，并为 IMErrno 全局变量赋值以示出错。

c) **名称**

statedraw(　)——显示状态信息。

格式

int statedraw(IM _ unit imp,ImStateInfo * status,caddr _ t udata);

说明

当状态区信息发生变化时调用此函数，显示状态信息。

imp　　指定输入对象。

status　　要显示的状态信息。

udata　　应用程序信息。

返回

返回 0 表示成功；返回—1 表示失败，并为 IMErrno 全局变量赋值以示出错。

d) **名称**

stateclear(　)——清除状态信息。

格式

int stateclear(IM _ unit imp,caddr _ t udata);

说明

清除状态区显示内容。

imp　　指定输入对象。

udata　　应用程序信息。

返回

返回 0 表示成功；返回—1 表示失败，并为 IMErrno 全局变量赋值以示出错。

e) **名称**

auxdraw(　)——显示选字区信息。

格式

int auxdraw(IM _ unit imp,IMAuxInfo * auxinfo,caddr _ t udata);

说明

显示选字区信息。

imp　　指定输入对象。
auxinfo　显示的选字区信息，存于 IMAuxInfo 结构之中。
udata　　应用程序信息。

返回

返回 0 表示成功；返回－1 表示失败，并为 IMErrno 全局变量赋值以示出错。

f) **名称**

auxclear( )——清除选字区信息。

格式

int auxclear(IM _ unit imp,caddr _ t udata);

说明

该函数清除输入对象的选字区。

imp　　指定输入对象。
udata　　应用程序信息。

返回

返回 0 表示成功；返回－1 表示失败，并为 IMErrno 全局变量赋值以示出错。

g) **名称**

imbeep( )——响铃。

格式

void imbeep( );

说明

通知应用程序响铃。

返回

无。

4.4.1.2　IM _ unit 数据结构

IM _ unit 数据结构是输入方法界面函数的公用部分数据结构，主要用于在系统或应用程序与输入方法模块之间交换数据。至少应包括下列元素：

int	imunit	公用信息。
IMTextInfo	* textinfo	预编辑区(正文区)信息。
IMAuxInfo	* auxinfo	选字区信息。
IMStateInfo	* stateinfo	状态区信息。
int	querystate	查找状态。
unsigned char	* convertstring	最终转换结果(汉字文件码，如 EUC 码)。
int	auxstate	选字区的状态。
IM _ cb	* imcb	辅助区函数入口表。

4.4.2　输出

4.4.2.1　显示

4.4.2.1.1　界面风格

计算机屏幕应分为显示主屏和输入提示屏。主屏幕的主要用户界面是 open、close、write 和 ioctl，这些系统调用完成主屏的打开、关闭、滚动、清除、定光标等功能。

输入提示屏和主屏采用统一界面。输入提示屏的用户界面完成提示屏的打开、关闭、清除、定光标等

功能。

4.4.2.1.2 字库、模式及工作方式的支持

a) **名称**

setdisplang——设置终端工作方式。

格式

int setdisplang(int fd,lang_t * lang)

说明

将终端工作方式设置为中文、英文方式或 UCS 方式等。

返回

0 成功;

−1 失败。

b) **名称**

getdisplang——获取终端工作方式。

格式

int getdisplang(int fd,lang_t * lang)

说明

获取终端工作方式。

返回

0 成功;

−1 失败。

c) **名称**

loaddispfont——装入指定字库。

格式

int loaddispfont(int fd,font_t * font)

说明

在字符发生器中装入指定字库。

返回

EFONTLOADED	字库已装入。
ENOMEM	无空间。
0	成功。

d) **名称**

unloaddispfont——拆卸指定字库。

格式

int unloaddispfont(int fd,font_t * font)

说明

拆卸字符发生器中字库。

返回

EFONTUNLOADED	字库未被装入。
EFONTBUSY	字库忙。
0	成功。

4.4.2.1.3 数据结构

```
#define        LG_FONT_MAX_NUM      10   /* 字库最大个数 */
```

```
struct    langmode{
    int   lg_type;        /*语言类型:
                          0:英文,1:中文(GB),...,
                          255:UCS(GB13000.1)*/
    int    lg_cardmode;       /*显示卡工作方式:英文或中文方式*/
    int    lg_workmode;       /*显示卡显示方式:正文或图形方式*/
    int    lg_dispmode;       /*显示模式*/
    int    lg_cardtype;       /*显示卡类型:
                          0:无效;
                          1:EGA;
                          2:VGA;
                          3:GW_CEGA;
                          4:GW_CVGA;
                          5:ERGO_CVGA;
                          6:LC_CEGA;
                          7:8900C TVGA;
                          …*/
};
typedef      struct langmode langmode_t;
struct    font{
    int    ft_id;             /*字模标识符:
                          0:无字库(默认);
                          1:16点阵GB1988(ASCII)软字库;
                          2:16点阵中文软字库;…*/
    int    ft_lib;            /*字库类型:
                          0:无字库(在英文方式下,因为VGA卡上有固化的GB1988(ASCII)字库而
                            无需装入字库);
                          1:软字库;
                          2:硬字库;
                          3:预加型硬字库。
                          */
    int    ft_font;       /*字体*/
    int    ft_dot;            /*字模类型:点阵或矢量,并可习惯分为:7,9,16,24,...*/
    int    ft_size;       /*字库大小*/
    char   *ft_buf;       /*字库缓冲区*/
    int    ft_usenum;         /*使用次数*/
    struct    font     *ft_next;/*指向下一个font结构*/
};
typedef      struct font font_t;

struct       lang{
    langmode_t    lg_mode;     /*终端工作状态*/
    int           lg_num;      /*字库个数*/
```

```
    font_t          lg_font[LG_FGONT_MAX_NUM];/*字库描述*/
};
typedef struct lang lang_t;
```

4.4.2.2 打印

4.4.2.2.1 中文打印界面

中文打印以 UNIX 打印机假脱机系统为基础，打印命令、打印机安装和打印队列管理与打印英文方式相同。

4.4.2.2.2 中文打印程序结构

中文打印程序分为中文打印处理程序和打印机驱动程序。

a) 中文打印处理程序

分析处理用户级打印控制命令，调用打印机驱动程序进行打印。

b) 打印机驱动程序

根据打印机控制命令文件(printcap)，驱动各种打印机进行打印输出。

4.4.2.2.3 中文打印控制命令

中文打印控制命令是打印中文所需的最基本控制命令集。打印控制命令以控制序列及控制功能给出。

控制序列的格式是：CSI P…PI…IF

其中：

CSI 为控制序列引导符，由 1/11 5/11(7 位环境)和 9/11(8 位环境)表示。

P 为参数，分为 Pn(数值参数)和 Ps(选择性参数)两种。

I 为中间字节，与终止字节一起标识控制功能。

F 为终止字节，它使控制序列终止，并与中间字节一起标识控制功能。

下面给出打印控制命令的详细定义：

a) **名称**

FNT——字体选择(font selection)。

表示

CSI Ps1;Ps2 02/00 04/04

默认参数值

Ps1=0;Ps2=0

说明

FNT 用于指出使用的字符字体。此字体由后面介绍的图形显现式样(SGR)选作基本字体或替换字体。

Ps1 指定基本字体或有关的替换字体：

0 基本字体——宋体；

1 第一替换字体——仿宋；

2 第二替换字体——黑体；

3 第三替换字体——楷体。

Ps2 是根据某个建立的暂存器来指定字符字体。其他字体从编号 12 开始定义。

1 标宋；

2 报宋；

3 大黑；

4 美黑；

5 行楷；

6 细圆；
7 准圆；
8 隶书；
9 隶变；
10 魏碑；
11 姚体。

b) **名称**

GSS——字符图形大小选择(graphic size selection)。

表示

CSI Pn 02/00 04/03

默认参数值

无。

说明

GSS 用于规定由字体选择(FNT)标明的基本字体和替换字体的高度和宽度。在 GSS 再次出现以前，所规定的值一直保持有效。

Pn 指定其高度，宽度由高度隐式规定。

参数值的单位由单位大小选择(SSU)规定。

c) **名称**

GSM——字符图形大小修改(graphic size modification)。

表示

CSI Pn1;Pn2 02/00 04/02

默认参数值

Pn1=100;Pn2=100

说明

GSM 用于修改由字符图形大小选择(GSS)规定的和由字体选择(FNT)标明的所有基本字体和替换字体的高度和(或)宽度。在 GSM 或 GSS 再次出现以前，所规定的值一直保持有效。

Pn1 指定其高度为 GSS 所规定的高度百分比。

Pn2 指定其宽度为 GSS 所规定的宽度百分比。

d) **名称**

SCS——置字间距(set character spacing)。

表示

CSI Pn 02/00 06/07

默认参数值

无。

说明

SCS 用于建立其后续文本的字间距。在 SCS、字间距选择(SHS)或间距增量(SPI)再次出现以前，所建立的间距一直保持有效。

Pn 指定字间距。

参数值的单位由单位大小选择(SSU)规定。

注：SHS 和 SPI 详见 GB/T 5261。本标准未作规定。

e) **名称**

SLS——置行间距(set line spacing)。

表示

CSI Pn 02/00 06/08

默认参数值

无。

说明

SCS 用于建立其后续文本的行间距。在 SLS、行间距选择(SVS)或间距增量(SPI)再次出现以前,所建立的间距一直保持有效。

Pn 指定行间距。

参数值的单位由单位大小选择(SSU)规定。

注:SVS 和 SPI 详见 GB/T 5261。本标准未作规定。

f) **名称**

SCO——选择字符旋转(select character orientation)。

表示

CSI Ps 02/00 06/05

默认参数值

Ps=0

说明

SCO 用于建立其后续图形字符的旋转量。在 SCO 再次出现以前,所建立的值一直保持有效。

其参数值是:

0	0°
1	45°
2	90°
3	135°
4	180°
5	225°
6	270°
7	315°

旋转是正的,也就是反时针方向,并适应于图形字符沿着字向方向正常呈现。受影响的图形字符的旋转中心本标准不作规定。

g) **名称**

SSU——单位大小选择(select size unit)。

表示

CSI Ps 02/00 04/09

默认参数值

Ps=0

说明

SSU 用于建立表示格式控制符功能的数值参数的单位。在 SSU 再次出现以前,所建立的单位一直保持有效。

其参数值是:

0	字符;
1	毫米;

2 计算机的 0.1 点:0.03528mm(25.4mm 的 1/720);
3 0.1 第道点:0.03759mm(10/266mm);
4 密耳:0.0254mm(25.4mm 的 1/1000);
5 基本测量单位(BMU):0.02117mm(25.4mm 的 1/1200);
6 微米:0.001mm;
7 象素:在设备中能指定的最小单位;
8 0.1 点:0.03514mm(35/996mm)。

h) **名称**

SGR——图形显现式样选择(select graphic rendition)。

表示

CSI Ps…06/13

默认参数值

Ps=0

说明

SGR 用于为其后续文本建立一种或多种图形显现式样。在 SGR 再次出现以前,所建立的式样一直保持有效,这要取决于图形显现组合模式(GRCM)的设置。每种图形显现式样由一个参数值指定:

0 默认显现式样,不管 GRCM 如何设置,都使前面出现的任何一个 SGR 的作用失效;
1 黑体字;
2 细体字;
3 斜体字;
4 单下划线;
7 反象;
9 划去(字符仍可识别,但标上被删除的记号);
10 基本字体;
11 第一替换字体;
12 第二替换字体;
13 第三替换字体;
14 第四替换字体;
15 第五替换字体;
16 第六替换字体;
17 第七替换字体;
18 第八替换字体;
19 第九替换字体;
20 花体(哥特体);
21 双下划线;
22 正常体(非黑体、非细线体);
23 非斜体、非花体;
24 无下划线(无单下划线,也无双下划线);
27 正象;
29 不划去;
30 黑色;
31 红色;

32 绿色；
33 黄色；
34 蓝色；
35 品红色；
36 青色；
37 白色；
39 默认颜色；
40 黑背景色；
41 红背景色；
42 绿背景色；
43 黄背景色；
44 蓝背景色；
45 品红背景色；
46 青背景色；
47 白背景色；
49 默认背景色；
51 加框架；
52 加包围圈；
53 上横线；
54 无框架、无包围圈；
55 无上横线；
60 单下划线或单右边线；
61 双下划线或双右边线；
62 单上横线或单左边线；
63 双上横线或双左边线；
64 着重号；
65 使由参数 60 至 64 和 66 至 69 建立的显现式样的作用作废；
66 单下划线或单左边线；
67 双下划线或双左边线；
68 单上横线或单右边线；
69 双上横线或双右边线；
101 表格符连接；
102 取消表格符连接。

注：GRCM 之规定详见 GB/T 5261，本标准未作规定。参数值 5、6、8、25、26、28、38、48、50、56～59 与打印无关，本标准未作规定。

i) **名称**

PLD——行部分正移(partial line forward)。

表示

08/11 或 ESC 04/11

默认参数值

无。

说明

PLD 使操作位置沿行方向移到部分偏移的虚拟行的相应字符位置。这个偏移量应足以使后

继字符映象成下标，直至后面第一次出现行部分反移(PLU)为止，或者若紧接着前面的字符是上标，则将其后面的字符图形恢复到操作行。

PLD 与 PLU 以外的其他格式控制符之间的相互作用本标准不作规定。

j) **名称**

PLU——行部分反移(partial line backward)。

表示

08/12 或 ESC 04/12

默认参数值

无。

说明

PLU 使操作位置沿行相反的方向移到部分偏移的虚拟行的相应字符位置。这个偏移量应足以使后继字符映象成上标，直至后面第一次出现行部分正移(PLD)为止，或者若紧接着前面的字符是下标，则将其后面的字符图形恢复到操作行。

PLU 与 PLD 以外的其他格式控制符之间的相互作用本标准不作规定。

k) **名称**

JFY——整版(justify)。

表示

CSI Ps…02/00 04/06

默认参数值

Ps=0

说明

JFY 用于指出图形字符串的起始，而这个图形字符串将按下列参数值指定的布局来整版：

0 不需整版，前面文本结束整版；

1 填词；

2 词间间隔；

3 字母间隔；

4 连字符；

5 行首对齐；

6 行首和行末之间居中；

7 行末对齐；

8 意大利连字符。

再次出现 JFY 指出被整版的图形字符串结束。

行首由置行首(SLH)的参数规定，行末由置行末(SLL)的参数规定。

l) **名称**

SLH——置行首(set line home)。

表示

CSI Pn 02/00 05/05

默认参数值。

无。

说明

SLH 用于在操作行及后续文本各行规定一个位置，在出现换行时操作位置移到该位置，这里 n 等于 Pn 的值。规定的位置称为行首位置，在 SLH 再次出现以前一直保持有效。

m) **名称**

SLL——置行末(set line limit)。

表示

CSI Pn 02/00 05/06

默认参数值。

无。

说明

SLL 用于在操作行及后续文本各行规定一个位置，在该位置之后操作位置将不发生隐式移动，这里 n 等于 Pn 的值。规定的位置称为行末位置，在 SLL 再次出现以前一直保持有效。

n) **名称**

PFS——页格式选择(page format selection)。

表示

CSI Ps 02/00 04/10

默认参数值

Ps=0

说明

PFS 根据纸的大小建立文本页面的映象有效区。页面由换页(FF)引入。在 PFS 再次出现以前，所建立的映象区一直保持有效。其参数值是：

0 纵向基本文本通信格式；
1 横向基本文本通信格式；
2 纵向基本 A4 格式；
3 横向基本 A4 格式；
4 纵向北美信件格式；
5 横向北美信件格式；
6 纵向扩充 A4 格式；
7 横向扩充 A4 格式；
8 纵向北美法定格式；
9 横向北美法定格式；
10 A4 短行格式；
11 A4 长行格式；
12 B5 短行格式；
13 B5 长行格式；
14 B4 短行格式；
15 B4 长行格式。

o) **名称**

FF——换页(form feed)。

表示

00/12

默认参数值

无。

说明

FF 使操作位置移到下一张表格或下一页预定行上的相应字符位置。

p) **名称**

SPL——置页界限(set page limit)。

表示

CSI Pn 02/00 06/10

默认参数值。

无。

说明

SPL 用于为当前页和后续页建立行位置，打印信息不得超出这一位置。此位置称页界限位置，在 SPL 再次出现以前一直保持有效。

q) **名称**

SPH——置页首(set page home)。

表示

CSI Pn 02/00 06/09

默认参数值

无。

说明

SPH 用于为当前页和后续页建立行位置，当换页(FF)出现时，操作位置移到该位置，这里 n 等于 Pn 的值。在该位置之前操作位置将不发生隐式移动。

r) **名称**

HVP——置字向和行向位置(character and line position)。

表示

CSI Pn1;Pn2 06/06

默认参数值

Pn1=1;Pn2=1

说明

HVP 使操作位置移到行向方向位置 n 处和字向方向位置 m 处，这里 n 等于 Pn1 的值，m 等于 Pn2 的值。

s) **名称**

SLE——语言环境选择(select language environment)。

表示

CSI Ps 02/04 07/00

默认参数值

Ps=0

说明

SLE 用于选择接收装置的语言系统或编码系统，Ps 参数的代表意义有：

0　预设语言模式；

1　打开英文语言模式；

2　关闭英文语言模式；

3　打开中文语言模式；

4　关闭中文语言模式。

此命令允许多国语言的组合应用，但多国语言间的编码结构则不在此标准规格中描述。

t) **名称**

BMP——点阵打印(bitmap dump transfer)。

表示

CSI Pn1;Pn2;Pn... 02/01 07/01

默认参数值

无。

说明

BMP 用于打印图形块。Pn1 表示横向宽度图点数，Pn2 表示纵向高度图点数。Pn 用于图形定义，每个 Pn 为一个八位图点的数值。点阵图形以由左而右的方式表示，逐次以先左后右，先上后下的排列方式传输，每列不满八位者右边补零。

u) **名称**

SFS——选择字符字号(select font size)。

表示

CSI Ps 02/00 02/02 07/01

默认参数值

Ps=0

说明

SFS 用于选择字符字号，Ps 参数的意义是：

Ps	字号	大 小 (300dpi)	大 小 (1400dpi)	行 距 (1400dpi)	字 距 (1400dpi)
0	七	24x24	116x116(58)	145	63
1	小六	25x25	120x120(60)	150	65
2	六	32x32	154x154(77)	193	84
3	小五	36x36	172x172(86)	215	93
4	五	40x40	192x192(96)	240	104
5	小四	48x48	230x230(115)	288	125
6	四	50x50	240x240(120)	300	130
7	三	60x60	288x288(144)	360	156
8	小二	68x68	326x326(163)	408	177
9	二	80x80	384x384(192)	480	208
10	一	100x100	480x480(240)	600	260
11	小初	120x120	576x576(288)	720	312
12	初	125x125	600x600(300)	750	325
13	小特	150x150	720x720(360)	900	390
14	特	175x175	800x800(400)	1050	433
15	特大	200x200	960x960(480)	1200	520

4.4.2.2.4 打印机控制命令

打印机控制命令是与打印机密切相关的打印控制命令集。打印机驱动程序通过处理这些打印机控制命令对各种打印机进行控制，从而实现中文打印。

打印机控制命令由 printcap 文件进行描述。printcap 文件的格式与终端描述文件 termcap 文件相似

4.4.2.2.4.1 打印机控制命令描述文件(printcap)格式

printcap 中的项由一些被','隔开的字段组成，第一项给出打印机的名称，名称间用'|'分隔。第二项开始为功能描述，格式为：

功能名=功能描述。

例：HPLaserJet|HPLaserJet4L，

sbim=\e*r0A，rbim=\e*rB，...

4.4.2.2.4.2 打印机控制命令描述

a) 打印机类型

功能名：type

说明：type＝1 为点阵打印机；type＝0 为激光或喷墨打印机。

b) 启动打印位映象图

功能名：sbim

c) 结束打印位映象图

功能名：rbim

d) 设置打印分辨率

功能名：dpi

e) 设置位映象图宽度

功能名：wid

f) 设置颜色平面

功能名：pln

g) 设置打印方向

功能名：dir

h) 输出位映象图

功能名：Zout

i) 移至位映象的下一行

功能名：Zz

4.5 实用程序

以下实用程序应能处理中文：

cat	串接并显示文件。
cut	剪下文件每一行中所选中的字段。
date	显示和设置日期。
df	报告空闲的磁盘块数和文件数。
echo	回应实参。
ed	正文编辑程序。
egrep	用完全正则表达式在文件中搜索某个模式。
ex	正文编辑程序。
fold	长行折叠。
grep	从文件中搜索模式。
head	显示文件的前面几行。
join	关系数据库运算程序。
lex	生成完成简单词法分析任务的程序。
logger	给系统日志添加登记项。
lp	向 LP 打印服务程序提出发送/取消的请求。
ls	列出目录的内容。
mail	收发邮件。
paste	合并若干文件的相同行或一个文件中连续相同的后几行。
pr	显示文件。
printf	显示格式化的输出。
sed	流编辑程序。
sh	标准 shell，作业控制和受限命令解释程序。
sort	排序和/或合并文件。

tail　显示文件的最后部分。

tr　翻译字符。

vi　基于 ex 的面向屏幕(直观)显示的编辑程序。

who　谁在系统中。

yacc　另一个编译程序的编译程序。

4.6　图形界面

4.6.1　文字处理图形界面

中文平台的图形界面应与硬件无关。

图形界面中文平台标准主要包括国际化和本地化两个方面，具体包括：

- 建立中文输入模块的摘挂方法和界面；
- 建立各种点阵汉字库、矢量轮廓汉字库、曲线轮廓汉字库的摘挂方法和界面；
- 建立基于图形界面的汉字信息处理函数。

4.6.1.1　国际化(Internationalization)

图形界面的国际化处理主要在三个层次上：即 X 字体服务器层(X Font Server)、库函数层(Libraries)及命令层(Commands)。

4.6.1.1.1　X 字体服务器

4.6.1.1.1.1　文字处理类型

X 字体服务器应能处理以下三种类型的文字：

点阵(dot—matrix)、矢量轮廓(Vector outline)及曲线轮廓(curve Outline)文字。

4.6.1.1.1.2　处理模式

a) 点阵类型文字：

1) X 字体服务器定义一种正文格式的点阵文字类型数据结构 bdf(Bitmap Distribution Format)作为面向用户的数据类型接口。

2) 定义一种二进制编码类型的点阵文字类型数据结构 pcf(Portable Compiled Format)，用以面向 X 字体服务器。

3) 提供 bdf 数据结构同 pcf 数据结构转换的命令，用以沟通用户与 X 字体服务器。

4) 提供点阵文字的显示函数。

b) 矢量轮廓类型文字

1) X 字体服务器定义一种正文格式的矢量轮廓文字类型数据结构 vdf(Vector outline Distribution Format)作为面向用户的数据类型接口

2) 定义一种二进制编码类型的矢量轮廓文字类型数据结构 vcf(Vector Compiled Format)，用以面向 X 字体服务器。

3) 提供 vdf 数据结构向 vcf 数据结构转换的命令。

4) 提供矢量轮廓类型文字的显示函数。

c) 曲线轮廓类型文字

1) X 字体服务器定义一种正文格式的曲线轮廓文字类型描述结构 cdf(Curve Outline Distribution Format)作为面向用户的数据类型接口。

2) 定义一种二进制编码类型的曲线轮廓文字类型数据结构 ccf(Curve Compiled Format)用以面向 X 字体服务器。

3) 提供 cdf 数据结构向 ccf 数据结构的转换命令。

4) 提供曲线轮廓类型文字的显示函数。

d) 在字库索引文件中，对三种文字类型有相应描述，以便服务器区分。

4.6.1.1.1.3　数据结构

a) 字库索引数据结构以正文格式反映文字类型及与服务器有关的数据信息，直接面向用户。

b) vdf 文件数据结构包括前导文件及矢量坐标数据。前导文件包括字库的族、形态、特征、类型等信息。其数据结构应遵循 GB/T 13846、GB/T 13847 和 GB/T 13848。

c) cdf 文件数据结构包括前导文件及曲线控制点数据。前导文件包括字库的族、形态、特征、类型等信息。

d) XScaledFontStruct 数据结构，用于反映矢量轮廓及曲线轮廓文字库的特征信息。

4.6.1.1.2 库函数

4.6.1.1.2.1 有关 X 字体服务器的函数

a) **名称**

XLoadScaledFont——把字体装入服务器。

格式

```
#include<chnfont.h>
Font XLoadScaledFont(displaxy,font_name)
Display * display
char * font_name
```

说明

它将可缩放字体装入服务器。字体不应与特定屏幕相关，一旦字体有效，就可把它设置在任意 GC 的 font 里，从而可以在随后的画图请求中使用它。

返回

该字体资源的 ID 值。

b) **名称**

XQueryScaledFont——字体查询。

格式

```
#include<chnfont.h>
XQueryScaledFont(displary,font_ID)
Display * display
int font_ID
```

说明

它能查询出装入字体服务器的字体并列出与该 font_ID 相关的字体信息。它返回与指定字体相关的 XScaledFontStruct 结构的指针；如还没有装载这个字体，则返回 NULL。

返回

一个指向 XScaledFontStruct 数据结构的指针。

c) **名称**

XLoadQueryScaledFont——调入并查询字体。

格式

```
#include<chnfont.h>
XLoadQueryScaledFont(display,name)
Display * display
char * name
```

说明

它同时执行 XLoadScaledFont 和 XQueryScaledFont。也就是它既打开指定字体，也返回指向 XScaledFontStruct 结构的指针。如字体不存在，返回 NULL。

返回

一个指向 XScaledFontStruct 数据结构的指针。

d) **名称**

XFreeScaledFont——释放字体。

格式

```
#include<chnfont.h>
XFreeScaledFont(display,font_struct)
Display * display;
XScaledFontStruct * font_struct;
```

说明

它释放为 XQueryScaledFont 或 XloadQueryScaledFont 填写的字体信息结构而分配的存储空间。

返回

成功返回 0,不成功返回—1。

4.6.1.1.2.2 文字操作类函数

a) **名称**

XDrawScaledWord——显示一个可缩放的文字。

格式

```
#include<chnfont.h>
XDrawScaledWord(display,drawable,gc,x,y,x_ratio,y_ratio,fill,style,code)
Display * displary;
Drawable drawable;
GC gc;
int     x,y;(文字基线的起始位置相对于指定可画物原点的 x、y 坐标)
int     x_ratio,y_ratio;(字体的横纵向比例)
int     fill;(填充/轮廓)
int     style;(文字风格)
int     code;(文字代码)
```

说明

此函数在用户定义环境下,根据其规定的起始位置、字体风格、填充格式等要求写出一个可缩放的文字,它作为一个在窗口或象素图中绘出文本的原语加在 xlib 库中,其字体由 XLoadScaledFont 或 XLoadQueryScaledFont 来指定。其中文字风格应包括正常、旋转、倾斜等。

返回

成功返回 0,不成功返回—1。

b) **名称**

XDrawScaledString——写一个大小可变的字串。

格式

```
#include<chnfont.h>
XDrawScaledString(display,drawable,gc,x,y,x_ratio,y_ratio,fill,style,string)
Display * display
Drawable drawable
GC,gc
int     x,y;(第一个文字基线的起始位置相对于指定可画物原点的 x、y 坐标)
```

int x _ ratio,y _ ratio;(字体的横纵向比例)
int fill;(填充与否)
int style;(文字风格)
char *string;(文字串)

说明

此函数在用户定义环境下,根据规定的起始位置、文字比例、字体风格、填充格式来写出一个可缩放的文字串,其字体由 XLoadScaledFont 或 XLoadQueryScaledFont 来定义。

返回

成功返回 0,不成功返回-1。

c) **名称**

XSetRotateAngle——设置文字的旋转(倾斜)角度。

格式

#include<chnfont.h>
XSetRotateAngle(angle);
Double angle;(文字旋转或倾斜角度)

说明

在调用 XDrawScaledWord 或 XDrawScaledString 函数时,若字体风格为旋转或倾斜效果,应在此之前调用该函数。

返回

成功返回 0,不成功返回-1。

4.6.1.1.3 命令

a) 工具类:

fsinfo:列字体服务器信息。

mkfontdir:创建字体索引文件。

xfd:列出字体。

xset:设置字体与服务器间联系。

b) 转换类:

bdftopcf:正文格式的点阵文字数据结构(bdf)向字体服务器可识别的二进制编码格式(pcf)转换。

cdftoccf:正文格式的曲线轮廓文字数据结构(cdf)向字体服务器可识别的二进制编码格式(ccf)转换。

gbtobdf:中华人民共和国国家标准点阵汉字格式(gb)向正文格式的点阵文字数据结构(bdf)转换。

vdftovcf:正文格式的矢量轮廓文字数据结构(vdf)向字体服务器可识别的二进制编码格式(vcf)转换。

4.6.1.2 本地化(Localization)

图形界面应提供以下应用程序:

a) 提供汉字造字工具;

b) 提供汉字图标编辑器;

c) 提供汉字图文编辑工具;

d) 提供汉字图文打印工具。

4.6.2 图形界面汉字输入机制

4.6.2.1 图形界面汉字输入服务器

汉字输入服务器按其功能可以分为输入机制管理层、输入方法处理层、辅助区处理层三个层次。

输入机制管理层:完成汉字输入服务器的初始化,装入指定的汉字输入方法,接收 X 服务器及应用

用户的汉字输入方法切换及调用请求，并负责向输入方法处理层解释传递输入请求，并将结果回送应用程序。

输入方法处理层：根据输入机制管理层的要求，按不同输入方法，自动查找码表，并回送查找结果。

辅助区处理层：提供汉字预编辑及选择提示的窗口显示。

4.6.2.2 图形界面汉字输入管理方式

汉字输入服务器为应用程序汉字输入请求提供服务，一个汉字输入服务器可以同时与本屏幕的多个客户进行互连。

4.6.2.3 图形界面汉字输入方法的摘挂

图形界面汉字输入方法应具有开放式的结构，用户能很方便地增加或删除。

a) 汉字输入环境设置

系统应为用户提供修改和设置汉字输入方法环境的命令，使汉字输入服务器方便地找到相应的汉字输入方法内部码表。

b) 汉字输入方法转换键定义

汉字输入方法转换键应以资源方式由用户定义，需要时由用户加入。

4.6.2.4 图形界面输入方法函数

a) **名称**

XOpenIM——在指定窗口内装入相应的输入方法模块，创建相应输入方法目标，并返回调用者。

格式

```
#indude<XIMlibint.h>

XIM XOpenIM(display,rdb,res_name,res_class)
Display * display;
XrmDatabase rdb;
Char * res_name;
Char * res_class;
```

说明

该函数为指定窗口装入指定的汉字输入方法。display 指定了需要装入输入方法的窗口，rdb 参数指向资源数据库，供输入方法查找相应资源，如果 rdb 为 NULL，则输入方法无资源可利用。res_name 和 res_class 参数指定了应用程序的资源名称和类型，它在输入方法查找公共资源时有用。

返回

成功，返回指向相应输入方法的目标指针；

失败，返回空 NULL。

b) **名称**

XCloseIM——关闭相应输入方法，并释放输入方法所占资源。

格式

```
#include<XIMlibint.h>
Status XCloseIM(supim)
XIM supim;
```

说明

关闭指定输入方法并释放该输入方法所占全部资源。

返回

Status XClose IM(supim)

XIM supim;

说明

关闭指定输入方法并释放该输入方法所占全部资源。

返回

返回当前输入方法管理模块状态。

c) **名称**

XGetIMValues——查询当前输入方法。

格式

#include<XIMlibint.h>

Char * XGetIMValues(supim,args)

XIM supim;

XIMArg * args;

说明

该函数查询当前输入方法是否具有指定参数设置。

返回

成功,返回 NULL;

失败,返回不满足的第一个参数名称。

d) **名称**

XDisplayOfIM——获取与输入方法相关联的窗口。

格式

Display * XDisplayOfIM(im)

XIM im;

说明

该函数用于获取与输入方法相关联的窗口。

返回

成功,返回窗口 ID 号;

失败,返回 NULL。

e) **名称**

XLocaleOfIM——获取与输入方法相关联的本地化语言。

格式

char * XLocaleOfIM(im)

XIM im;

说明

该函数可用于检查系统对指定的本地化语言是否支持。

返回

成功,返回与输入方法相关联的本地化语言;

失败,返回 NULL。

f) **名称**

XCreateIC——创建指定输入方法的输入上下文对象,并申请相应资源。

格式

include<Xlibint.h>

XIC XCreateIC (im,va _ alist)

XIM im;

Va_dcl va_alist;

说明

创建一个输入方法上下文时可以指定参数要求，如输入风格以及返回值集合等；如果这些参数要求不能设置，则失败返回。

失败原因如下：

1）必需的参数未被设置；

2）试图设置只读参数；

3）参数名称有错；

4）输入方法执行有错。

返回

成功，返回相应输入上下文对象句柄；

失败，返回 NULL。

g）**名称**

XDestroyIC——关闭指定输入方法的输入上下文对象并释放相应资源。

格式

#include<Xlibint.h>

void XDestroyIC(ic)

XIC ic;

说明

该函数关闭指定输入方法的输入上下文对象并释放相应资源。

返回

无。

h）**名称**

XSetICFocus——设置聚焦窗口的输入上下文。

格式

#include<XLcint.h>

void XSetICFocus(ic)

XIC ic;

说明

该函数允许所有键盘或鼠标的汉字输入请求均集中在聚焦窗口上，输入方法的反馈响应均指向该窗口

返回

无。

i）**名称**

XUnsetICFocus——释放聚焦窗口的输入上下文。

格式

#include<XLcint.h>

void XUnsetICFocus(ic)

XIC ic;

说明

释放聚焦窗口的输入上下文，以后输入方法的所有反馈响应与此窗口无关。

返回

无。

j) **名称**

XwcResetIC——重新设置输入方法上下文为初始化状态。

格式

```
char * XwcResetIC(ic)
XIC ic;
```

说明

该函数重新设置输入方法上下文为初始化状态，此时与该输入方法上下文相关的所有输入均被丢弃，预编辑区域被清除，但聚焦状态不变。

返回

成功，返回预编辑字串；

失败，返回 NULL。

k) **名称**

XIMOfIC——获取与输入方法上下文相关联的输入方法。

格式

```
XIM XIMOfIC(ic)
XIC ic;
```

说明

该函数用于获取与输入方法上下文相关联的输入方法。

返回

成功，返回输入方法；

失败，返回 NULL。

l) **名称**

XGetICValues——获取当前输入方法的输入上下文的参数设置情况。

格式

```
#include<XLcint.h>
Char * XGetICValues(ic,va _ alist)
XIC ic;
Va _ dcl va _ alist;
```

说明

失败原因：

1) 参数名称有误；

2) 输入方法执行有误。

返回

成功，返回 NULL；

失败，返回第一个出错参数名称。

m) **名称**

XSetICValues——设置输入上下文的参数。

格式

```
#include<XLcint.h>
Char * XSetICValues(ic,va _ alist)
XIC ic;
Va _ dcl va _ alist;
```

说明

以下原因失败返回：

1）试图设置只读参数；

2）参数名称有错；

3）输入方法执行出错。

返回

成功，返回 NULL；

失败，返回出错的第一个参数名称。

n）**名称**

GeometryCallback——设置预显窗口的几何参数。

格式

void GeometryCallback(ic,client _ data,call _ data)

XIC ic;

XPointer client _ data;

XPointer call _ data;

说明

此函数 call _ data 必须设为 NULL。

返回

无。

o）**名称**

PreeditStartCallback——初始化预编辑。

格式

int PreeditStartCallback(ic,client _ data,call _ data)

XIC ic;

XPointer client _ data;

XPointer call _ data;

说明

当输入方法的输入转换置 ON 时，调用此函数初始化预编辑区域。其中，

call _ data 必须设为 NULL。

返回

预编辑区域最大的预编辑字串长度，如大于零，则表示其为最大长度；如返回为－1，则表示无长度限制。

p）**名称**

PreeditDoneCallback——清除预编辑区域。

格式

void PreeditDoneCallback(ic,client _ data,call _ data)

XIC ic;

XPointer client _ data;

XPointer call _ data;

说明

当输入方法关闭输入方法转换时，调用此函数清除预编辑区域。如果已指定了某一个输入方法上下文，则 call _ data 必须置 NULL。

返回

无。

q) **名称**

PreeditDrawCallback——编辑、插入、删除或替换预编辑区域、预编辑字串。

格式

```
void PreeditDrawCallback(ic,client _ data,call _ data)
XIC ic;
XPointer client _ data;
XPointer call _ data;
```

说明

该函数中,call _ data 含有预编辑字串信息,以结构方式传递,并且相应字串被显示在预编辑窗口后,光标也做相应移动。

返回

无。

r) **名称**

PreeditCaretCallback——设置输入字串的插入点。

格式

```
void PreeditCaretCallback(ic,client _ data,call _ data)
XIC ic;
XPointer client _ data;
XPointer call _ data;
```

说明

输入方法应设有一些特殊键允许用户自己去移动预编辑区光标位置。能插入字符操作等。其中 call _ data 指明预编辑光标移动信息。

返回

无。

s) **名称**

StatusStartCallback——设置输入方法上下文为启动状态。

格式

```
void StatusStartCallback(ic,client _ data,call _ data)
XIC ic;
XPointer client _ data;
XPointer call _ data;
```

说明

当一个输入方法上下文被创建并聚焦后,调用该函数,设置初始的显示状态,等待显示输入。其中参数 call _ data 为 NULL。

返回

空。

t) **名称**

StatusDoneCallback——设置输入方法上下文为空状态。

格式

```
void StatusDoneCallback(ic,client _ data,call _ data)
XIC ic;
XPointer client _ data;
```

XPointer call _ data;

说明

当一个输入方法上下文被删除，并且聚焦被释放后，调用该函数，其中 call _ data 为 NULL。此时释放所有相关数据。

返回

无。

u) **名称**

StatusDrawCallback——修改输入方法上下文状态。

格式

void StatusDrawCallback(ic,client _ data,call _ data)

XIC ic;

XPointer client _ data;

XPointer call _ data;

说明

当输入方法上下文状态改变时调用此函数。由 call _ data 指定状态改变信息。

返回

无。

v) **名称**

XwcLookupString——从输入方法中获取键盘输入的结果。

格式

int XwcLookupString (ic, event, buffer _ return, bytes _ buffer, keysym _ return, status _ return)

XIC ic;

XKeyPressedEvent * event;

char * buffer _ return;

int bytes _ buffer;

Keysym * keysym _ return;

Status * status _ return;

说明

该函数从输入方法中获取键盘输入的结果，包括：

ic	指定的输入方法上下文。
event	键盘事件。
buffer _ return	从输入方法中返回的宽字符字串。
bytes _ buffer	返回字串内有效的字节数。

返回

从输入方法返回 buffer _ return 的字节数。

4.6.2.5 数据结构

a) XIMPreeditDrawCallback Struct 数据结构

int caret;光标偏移量

int chg _ first;开始变化位置

int chg _ length;变化的字符个数

XIMText * text;变化的字串

b) XIMPreeditCaretCallback Struct 数据结构

int position；光标位置

XIMCaretDirection direction；光标移动方向

XIMCaretStyle Style；光标移动方式

附 录 A
（标准的附录）
POSIX 中文特征文件

A1 范围

POSIX 中文特征文件是定义使用GB/T 14246.1(以下简称 POSIX.1)的中文环境。根据 POSIX 地区规范指南的要求，本规范给出了某些参数和选项的具体技术规格，以支持对与中国语言文化密切相关的事务进行处理。

本文件的目标是：

a) 确定满足中文环境需要、并能促进开发下述程序的一组最小最合适的技术规格：

1) 遵循 POSIX 的系统实现；

2) 可移植的应用程序开发。

b) 进一步定义下列参数在 POSIX.1 和 ISO/IEC 9945.2(以下简称 POSIX.2)中的技术规格：

1) 参数；

2) 选项。

A2 遵循度

遵循 POSIX 的中文系统的具体实现应该支持本文件确定的全部参数和选项。

遵循 POSIX 的系统实现并不一定遵循 POSIX 中文特征文件，所以能通过 POSIX 的系统实现不一定能通过 POSIX 中文特征文件的规范。

一个遵循 POSIX 的中文系统应支持 GB/T 15272。

A3 参数和选项

Charmap 字符映射表

此字符映射表是以 GB 2312 为基础的。

CHAR BIT and Character Handling in C Language 字符比特数及 C 语言的字符处理

POSIX 中文依从系统要求字符比特数 CHAR BIT 为 8。这就是说，按 GB/T 15272 扩展字符集的概念，字符集被定义为非单字节的，即 MB_CUR_MAX 值大于等于 2。

Delete Character 字符删除

有一点很重要，即使是对多字节字符的删除操作，也应该与正常操作完全一样。这就是说，如果有一中文字符，那么，删除该字符的输入，像 POSIX.1 中定义的那样，就是删除该中文字符。

File Name Length and its Handling 文件名长度及其处理

POSIX 中文依从系统要求：在多字节字符处理过程中，当文件名长度超过 NAME_MAX 时，应将 POSIX_NO_TRUNC 变量置成真，使得在不截短文件名字符串的情况下报告 ENAMETOOLONG 错误。

如果要求内核检查字符边界来代替提供 ENAMETOOLONG 出错报告，那是困难而效率低下的，因为混在串中的字符可能遗漏。确定最大文件名长度 NAME_MAX 值时应考虑多字节字符的编码长度。

Character Encoding for Path Name Delimiters 路径名定界符的字符编码

内核必须检测路径名中的斜杠字符(/)和句号字符(.)。上述字符既可以是按 GB 1988 规定的单字节字符编码，也可以是按 GB 2312 规定的多字节的中文字符编码。

A4 POSIX 中文特征文件

下面的字符映射表是基于 GB 2312 的。
本字符映射表包括 GB 1988 中的 128 个字符以及 GB 2312 字符集里的全部字符。

CHARMAP

<code_set_name>	GB 2312
<mb_cur_max>	2
<mb_cur_min>	1
<NULL>	\x00
<SOH>	\x01
<STX>	\x02
<ETX>	\x03
<EOT>	\x04
<ENQ>	\x05
<ACK>	\x06
<alert>	\x07
<backspace>	\x08
<tab>	\x09
<newline>	\x0A
<vertical-tab>	\x0B
<form-feed>	\x0C
<carriage-return>	\x0D
<SO>	\x0E
<SI>	\x0F
<DLE>	\x10
<DC1>	\x11
<DC2>	\x12
<DC3>	\x13
<DC4>	\x14
<NAK>	\x15
<SYN>	\x16
<ETB>	\x17
<CAN>	\x18
<EM>	\x19
<SUB>	\x1A
<ESC>	\x1B
<IS4>	\x1C
<IS3>	\x1D
<IS2>	\x1E
<IS1>	\x1F

<space>	\x20
<exclamation mark>	\x21
<quotation—mark>	\x22
<number—sign>	\x23
<dollar—sign>	\x24
<percent—sign>	\x25
<ampersand>	\x26
<apostrophe>	\x27
<left—parenthesis>	\x28
<right—parenthesis>	\x29
<asterisk>	\x2A
<plus—sign>	\x2B
<comma>	\x2C
<hyphen>	\x2D
<hyphen—minus>	\x2D
<period>	\x2E
<slash>	\x2F
<zero>	\x30
<one>	\x31
<two>	\x32
<three>	\x33
<four>	\x34
<five>	\x35
<six>	\x36
<seven>	\x37
<eight>	\x38
<nine>	\x39
<colon>	\x3A
<semicolon>	\x3B
<less—than—sign>	\x3C
<equals—sign>	\x3D
<greater—than—sign>	\x3E
<question—mark>	\x3F
<commercial—at>	\x40
<A>	\x41
<B>	\x42
<C>	\x43
<D>	\x44
<E>	\x45
<F>	\x46
<G>	\x47
<H>	\x48
<I>	\x49

<J>	\x4A
<K>	\x4B
<L>	\x4C
<M>	\x4D
<N>	\x4E
<O>	\x4F
<P>	\x50
<Q>	\x51
<R>	\x52
<S>	\x53
<T>	\x54
<U>	\x55
<V>	\x56
<W>	\x57
<X>	\x58
<Y>	\x59
<Z>	\x5A
<left—square—bracket>	\x5B
<backslash>	\x5C
<reverse—solidus>	\x5C
<right—square—bracket>	\x5D
<circumflex>	\x5E
<circumflex—accent>	\x5E
<underscore>	\x5F
<low—line>	\x5F
<grave—accent>	\x60
<a>	\x61
<b>	\x62
<c>	\x63
<d>	\x64
<e>	\x65
<f>	\x66
<g>	\x67
<h>	\x68
<i>	\x69
<j>	\x6A
<k>	\x6B
<l>	\x6C
<m>	\x6D
<n>	\x6E
<o>	\x6F
<p>	\x70
<q>	\x71

<r>	\x72
<s>	\x73
<t>	\x74
<u>	\x75
<v>	\x76
<w>	\x77
<x>	\x78
<y>	\x79
<z>	\x7A
<left－brace>	\x7B
<left－curly－bracket>	\x7B
<vertical－line>	\x7C
<right－brace>	\x7D
<right－curly－bracket>	\x7D
<tilde>	\x7E
<DEL>	\x7F

字符的名称根据它们的区位码(十进制)给出,在区位码前加 GB 字样。

<GB01－01>…<GB01—94>	\xA1\xA1
<GB02－17>…<GB02—66>	\xA2\xB1
<GB02－69>…<GB02—78>	\xA2\xE5
<GB02－81>…<GB02—92>	\xA2\xF1
<GB03－01>…<GB03—94>	\xA3\xA1
<GB04－01>…<GB04—83>	\xA4\xA1
<GB05－01>…<GB05—86>	\xA5\xA1
<GB06－01>…<GB06—24>	\xA6\xA1
<GB06－33>…<GB06—56>	\xA6\xC1
<GB07－01>…<GB07—33>	\xA7\xA1
<GB07－49>…<GB07—81>	\xA7\xD1
<GB08－01>…<GB08—26>	\xA8\xA1
<GB08－37>…<GB08—73>	\xA8\xC5
<GB09－04>…<GB09—79>	\xA9\xA4
# 一级汉字：	
<GB16－01>…<GB16—94>	\xB0\xA1
<GB17－01>…<GB17—94>	\xB1\xA1
<GB18－01>…<GB18—94>	\xB2\xA1
<GB19－01>…<GB19—94>	\xB3\xA1
<GB20－01>…<GB20—94>	\xB4\xA1
<GB21－01>…<GB21—94>	\xB5\xA1
<GB22－01>…<GB22—94>	\xB6\xA1
<GB23－01>…<GB23—94>	\xB7\xA1
<GB24－01>…<GB24—94>	\xB8\xA1

<GB25—01>…<GB25—94> \xB9\xA1
<GB26—01>…<GB26—94> \xBA\xA1
<GB27—01>…<GB27—94> \xBB\xA1
<GB28—01>…<GB28—94> \xBC\xA1
<GB29—01>…<GB29—94> \xBD\xA1
<GB30—01>…<GB30—94> \xBE\xA1
<GB31—01>…<GB31—94> \xBF\xA1
<GB32—01>…<GB32—94> \xC0\xA1
<GB33—01>…<GB33—94> \xC1\xA1
<GB34—01>…<GB34—94> \xC2\xA1
<GB35—01>…<GB35—94> \xC3\xA1
<GB36—01>…<GB36—94> \xC4\xA1
<GB37—01>…<GB37—94> \xC5\xA1
<GB38—01>…<GB38—94> \xC6\xA1
<GB39—01>…<GB39—94> \xC7\xA1
<GB40—01>…<GB40—94> \xC8\xA1
<GB41—01>…<GB41—94> \xC9\xA1
<GB42—01>…<GB42—94> \xCA\xA1
<GB43—01>…<GB43—94> \xCB\xA1
<GB44—01>…<GB44—94> \xCC\xA1
<GB45—01>…<GB45—94> \xCD\xA1
<GB46—01>…<GB46—94> \xCE\xA1
<GB47—01>…<GB47—94> \xCF\xA1
<GB48—01>…<GB48—94> \xD0\xA1
<GB49—01>…<GB49—94> \xD1\xA1
<GB50—01>…<GB50—94> \xD2\xA1
<GB51—01>…<GB51—94> \xD3\xA1
<GB52—01>…<GB52—94> \xD4\xA1
<GB53—01>…<GB53—94> \xD5\xA1
<GB54—01>…<GB54—94> \xD6\xA1
<GB55—01>…<GB55—89> \xD7\xA1

二级汉字：

<GB56—01>…<GB56—94> \xD8\xA1
<GB57—01>…<GB57—94> \xD9\xA1
<GB58—01>…<GB58—94> \xDA\xA1
<GB59—01>…<GB59—94> \xDB\xA1
<GB60—01>…<GB60—94> \xDC\xA1
<GB61—01>…<GB61—94> \xDD\xA1
<GB62—01>…<GB62—94> \xDE\xA1
<GB63—01>…<GB63—94> \xDF\xA1
<GB64—01>…<GB64—94> \xE0\xA1
<GB65—01>…<GB65—94> \xE1\xA1
<GB66—01>…<GB66—94> \xE2\xA1

<GB67 01>…<GB67 94> \xE3\xA1
<GB68 01>…<GB68 94> \xE4\xA1
<GB69 01>…<GB69 94> \xE5\xA1
<GB70 01>…<GB70 94> \xE6\xA1
<GB71 01>…<GB71 94> \xE7\xA1
<GB72 01>…<GB72 94> \xE8\xA1
<GB73 01>…<GB73 94> \xE9\xA1
<GB74 01>…<GB74 94> \xEA\xA1
<GB75 01>…<GB75 94> \xEB\xA1
<GB76 01>…<GB76 94> \xEC\xA1
<GB77 01>…<GB77 94> \xED\xA1
<GB78 01>…<GB78 94> \xEE\xA1
<GB79 01>…<GB79 94> \xEF\xA1
<GB80 01>…<GB80 94> \xF0\xA1
<GB81 01>…<GB81 94> \xF1\xA1
<GB82 01>…<GB82 94> \xF2\xA1
<GB83 01>…<GB83 94> \xF3\xA1
<GB84 01>…<GB84 94> \xF4\xA1
<GB85 01>…<GB85 94> \xF5\xA1
<GB86 01>…<GB86 94> \xF6\xA1
<GB87 01>…<GB87 94> \xF7\xA1

END CHARMAP

LC_CTYPE

大小写包括：
GB 1988 中的字母。
GB 2312 中的拉丁字母。
GB 2312 中的希腊字母。
GB 2312 中的俄文字母。

upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;
<M>;\
<N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;
<Z>;\
<GB03 33>;…;<GB03 58>;\
<GB06 01>;…;<GB06 24>;\
<GB07 01>;…;<GB07 33>
lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
<n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>;\
<GB03 65>;…;<GB03 90>;\
<GB06 33>;…;<GB06 56>;\

```
        <GB07—49>;…;<GB07—81>
digit   <zero>;<one>;<two>;<three>;<four>;\
        <five>;<six>;<seven>;<eight>;<nine>
space   <tab>;<newline>;<vertical—tab>;<form—feed>;<carriage—return>;\
        <space>;<GB01—01>
cntrl   <alert>;<backspace>;<tab>;<newline>;<vertical—tab>;\
        <form—feed>;<carriage—return>;\
        <NULL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
        <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
        <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
        <IS1>;<DEL>
punct   <exclamation—mark>;<quotation—mark>;<number—sign>;\
        <dollar—sign>;<percent—sign>;<ampersand>;<apostrophe>;\
        <left—parenthesis>;<right—parenthesis>;<asterisk>;\
        <plus—sign>;<comma>;<hyphen>;<period>;<slash>;\
        <colon>;<semicolon>;<less—than—sign>;<equals—sign>;\
        <greater—than—sign>;<question—mark>;<commercial—at>;\
        <left—square—bracket>;<backslash>;<right—square—bracket>;\
        <circumflex>;<underscore>;<grave—accent>;\
        <left—curly—bracket>;<vertical—line>;<right—curly—bracket>;<tilde>;\
        <GB01—02>;…;<GB01—94>;\
        <GB02—17>;…;<GB02—66>;\
        <GB02—69>;…;<GB02—78>;\
        <GB02—81>;…;<GB02—92>;\
        <GB03—01>;…;<GB03—15>;\
        <GB03—26>;…;<GB03—32>;\
        <GB03—59>;…;<GB03—64>;\
        <GB03—91>;…;<GB03—94>;\
        <GB09—04>;…;<GB09—79>
xdigit  <one>;<two>;<three>;<four>;<five>;<six>;<seven>;<eight>;<nine>;\
        <A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
blank   <space>;<tab>;<GB01—01>
fphonogram  <GB08—01>;…;<GB08—26>;\
            <GB08—37>;…;<GB08—73>
toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);
        (<f>,<F>);\
        (<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);(<k>,<K>);
        (<l>,<L>);\
        (<m>,<M>);(<n>,<N>);(<o>,<O>);(<p>,<P>);(<q>,<Q>);
        (<r>,<R>);\
        (<s>,<S>);(<t>,<T>);(<u>,<U>);(<v>,<V>);(<w>,<W>);
        (<x>,<X>);\
        (<y>,<Y>);(<z>,<Z>);\
```

```
            (<GB03—65>,<GB03—33>);…;(<GB03—90>,<GB03—58>);\
            (<GB06—33>,<GB06—01>);…;(<GB06—56>,<GB06—24>);\
            (<GB07—49>,<GB07—01>);…;(<GB07—81>,<GB07—33>)
tolower     (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);
            (<F>,<f>);\
            (<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);(<K>,<k>);
            (<L>,<l>);\
            (<M>,<m>);(<N>,<n>);(<O>,<o>);(<P>,<p>);(<Q>,<q>);
            (<R>,<r>);\
            (<S>,<s>);(<T>,<t>);(<U>,<u>);(<V>,<v>);(<W>,<w>);
            (<X>,<x>);\
            (<Y>,<y>);(<Z>,<z>);\
            (<GB03—33>,<GB03—65>);…;(<GB03—58>,<GB03—90>);\
            (<GB06—01>,<GB06—33>);…;(<GB06—24>,<GB06—56>);\
            (<GB07—01>,<GB07—49>);…;(<GB07—33>,<GB07—81>)
#   多字节字符(倍宽显示):

fullc       <GB01—01>;<GB01—71>;<GB03—01>;<GB03—02>;<GB03—03>;\
            <GB03—05>;…;<GB03—94>
undefchar   <GB01—94>

#   多字节字符到单字节字符的变换:

fctohc  (<GB01—01>,<space>);(<GB01—71>,<dollar—sign>);\
        (<GB03—01>,<exclamation—mark>);(<GB03—02>,<quotation—mark>);\
        (<GB03—03>,<number—sign>);\
        (<GB03—05>,<percent—sign>);…;(<GB03—94>,<tilde>)
hctofc    (<space>,<GB01—01>);(<dollar—sign>,<GB01—71>);\
          (<exclamation—mark>,<GB03—01>);(<quotation—mark>,<GB03—02>);\
          (<number—sign>,<GB03—03>);\
          (<percent—sign>,<GB03—05>);…;(<tilde>,<GB03—94>)

#   可打印字符:
#   print=alnum+punct+space+Chinese+other printable 2—byte chars

print       <exclamation—mark>;<quotation—mark>;<number—sign>;\
            <dollar—sign>;<percent—sign>;<ampersand>;<apostrophe>;\
            <left—parenthesis>;<right—parenthesis>;<asterisk>;\
            <plus—sign>;<comma>;<hyphen>;<period>;<slash>;\
            <colon>;<semicolon>;<less—than—sign>;<equals—sign>;\
            <greater—than—sign>;<question—mark>;<commercial—at>;\
            <left—square—bracket>;<backslash>;<right—square—bracket>;\
            <circumflex>;<underscore>;<grave—accent>;\
```

<left-curly-bracket>;<vertical-line>;<right-curly-bracket>;<tilde>;\
<A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;
<M>;\
<N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;
<Z>;\
<GB03—33>;…;<GB03—58;>;\
<GB06—01>;…;<GB06—24;>;\
<GB07—01>;…;<GB07—33;>;\
<a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
<n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>;\
<GB03—65>;…;<GB03—90>;\
<GB06—33>;…;<GB06—56>;\
<GB07—49>;…;<GB07—81>;\
<zero>;<one>;<two>;<three>;<four>;\
<five>;<six>;<seven>;<eight>;<nine>;\
<tab>;<newline>;<vertical-tab>;<form-feed>;<carriage-return>;\
<space>;<GB01—01>;\
<GB01—01>;…;<GB01—94>;\
<GB02—17>;…;<GB02—66>;\
<GB02—69>;…;<GB02—78>;\
<GB02—81>;…;<GB02—92>;\
<GB03—01>;…;<GB03—94>;\
<GB04—01>;…;<GB04—83>;\
<GB05—01>;…;<GB05—86>;\
<GB06—01>;…;<GB06—24>;\
<GB06—33>;…;<GB06—56>;\
<GB07—01>;…;<GB07—33>;\
<GB07—49>;…;<GB07—81>;\
<GB08—01>;…;<GB08—26>;\
<GB08—37>;…;<GB08—73>;\
<GB09—04>;…;<GB09—79>;\
<GB16—01>;…;<GB16—94>;\
<GB17—01>;…;<GB17—94>;\
<GB18—01>;…;<GB18—94>;\
<GB19—01>;…;<GB19—94>;\
<GB20—01>;…;<GB20—94>;\
<GB21—01>;…;<GB21—94>;\
<GB22—01>;…;<GB22—94>;\
<GB23—01>;…;<GB23—94>;\
<GB24—01>;…;<GB24—94>;\
<GB25—01>;…;<GB25—94>;\
<GB26—01>;…;<GB26—94>;\
<GB27—01>;…;<GB27—94>;\

<GB28—01>;…;<GB28—94>;\
<GB29—01>;…;<GB29—94>;\
<GB30—01>;…;<GB30—94>;\
<GB31—01>;…;<GB31—94>;\
<GB32—01>;…;<GB32—94>;\
<GB33—01>;…;<GB33—94>;\
<GB34—01>;…;<GB34—94>;\
<GB35—01>;…;<GB35—94>;\
<GB36—01>;…;<GB36—94>;\
<GB37—01>;…;<GB37—94>;\
<GB38—01>;…;<GB38—94>;\
<GB39—01>;…;<GB39—94>;\
<GB40—01>;…;<GB40—94>;\
<GB41—01>;…;<GB41—94>;\
<GB42—01>;…;<GB42—94>;\
<GB43—01>;…;<GB43—94>;\
<GB44—01>;…;<GB44—94>;\
<GB45—01>;…;<GB45—94>;\
<GB46—01>;…;<GB46—94>;\
<GB47—01>;…;<GB47—94>;\
<GB48—01>;…;<GB48—94>;\
<GB49—01>;…;<GB49—94>;\
<GB50—01>;…;<GB50—94>;\
<GB51—01>;…;<GB51—94>;\
<GB52—01>;…;<GB52—94>;\
<GB53—01>;…;<GB53—94>;\
<GB54—01>;…;<GB54—94>;\
<GB55—01>;…;<GB55—89>;\
<GB56—01>;…;<GB56—94>;\
<GB57—01>;…;<GB57—94>;\
<GB58—01>;…;<GB58—94>;\
<GB59—01>;…;<GB59—94>;\
<GB60—01>;…;<GB60—94>;\
<GB61—01>;…;<GB61—94>;\
<GB62—01>;…;<GB62—94>;\
<GB63—01>;…;<GB63—94>;\
<GB64—01>;…;<GB64—94>;\
<GB65—01>;…;<GB65—94>;\
<GB66—01>;…;<GB66—94>;\
<GB67—01>;…;<GB67—94>;\
<GB68—01>;…;<GB68—94>;\
<GB69—01>;…;<GB69—94>;\
<GB70—01>;…;<GB70—94>;\

<GB71—01>;…;<GB71—94>;\
<GB72—01>;…;<GB72—94>;\
<GB73—01>;…;<GB73—94>;\
<GB74—01>;…;<GB74—94>;\
<GB75—01>;…;<GB75—94>;\
<GB76—01>;…;<GB76—94>;\
<GB77—01>;…;<GB77—94>;\
<GB78—01>;…;<GB78—94>;\
<GB79—01>;…;<GB79—94>;\
<GB80—01>;…;<GB80—94>;\
<GB81—01>;…;<GB81—94>;\
<GB82—01>;…;<GB82—94>;\
<GB83—01>;…;<GB83—94>;\
<GB84—01>;…;<GB84—94>;\
<GB85—01>;…;<GB85—94>;\
<GB86—01>;…;<GB86—94>;\
<GB87—01>;…;<GB87—94>

图符=字母数字+标点符号+汉字+其他可打印的双字节字符

graph <exclamation—mark>;<quotation—mark>;<number—sign>;\
<dollar—sign>;<percent—sign>;<ampersand>;<apostrophe>;\
<left—parenthesis>;<right—parenthesis>;<asterisk>;\
<plus—sign>;<comma>;<hyphen>;<period>;<slash>;\
<colon>;<semicolon>;<less—than—sign>;<equals—sign>;\
<greater—than—sign>;<question—mark>;<commercial—at>;\
<left—square—bracket>;<backslash>;<right—square—bracket>;\
<circumflex>;<underscore>;<grave—accent>;\
<left—curly—bracket>;<vertical—line>;<right—curly—bracket>;<tilde>;\
<A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;
<M>;\
<N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;
<Z>;\
<GB03—33>;…;<GB03—58;>;\
<GB06—01>;…;<GB06—24;>;\
<GB07—01>;…;<GB07—33;>;\
<a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
<n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>;\
<GB03—65>;…;<GB03—90;>;\
<GB06—33>;…;<GB06—56;>;\
<GB07—49>;…;<GB07—81;>;\
<zero>;<one>;<two>;<three>;<four>;\
<five>;<six>;<seven>;<eight>;<nine>;\

<GB01—01>;…;<GB01—94>;\
<GB02—17>;…;<GB02—66>;\
<GB02—69>;…;<GB02—78>;\
<GB02—81>;…;<GB02—92>;\
<GB03—01>;…;<GB03—94>;\
<GB04—01>;…;<GB04—83>;\
<GB05—01>;…;<GB05—86>;\
<GB06—01>;…;<GB06—24>;\
<GB06—33>;…;<GB06—56>;\
<GB07—01>;…;<GB07—33>;\
<GB07—49>;…;<GB07—81>;\
<GB08—01>;…;<GB08—26>;\
<GB08—37>;…;<GB08—73>;\
<GB09—04>;…;<GB09—79>;\
<GB16—01>;…;<GB16—94>;\
<GB17—01>;…;<GB17—94>;\
<GB18—01>;…;<GB18—94>;\
<GB19—01>;…;<GB19—94>;\
<GB20—01>;…;<GB20—94>;\
<GB21—01>;…;<GB21—94>;\
<GB22—01>;…;<GB22—94>;\
<GB23—01>;…;<GB23—94>;\
<GB24—01>;…;<GB24—94>;\
<GB25—01>;…;<GB25—94>;\
<GB26—01>;…;<GB26—94>;\
<GB27—01>;…;<GB27—94>;\
<GB28—01>;…;<GB28—94>;\
<GB29—01>;…;<GB29—94>;\
<GB30—01>;…;<GB30—94>;\
<GB31—01>;…;<GB31—94>;\
<GB32—01>;…;<GB32—94>;\
<GB33—01>;…;<GB33—94>;\
<GB34—01>;…;<GB34—94>;\
<GB35—01>;…;<GB35—94>;\
<GB36—01>;…;<GB36—94>;\
<GB37—01>;…;<GB37—94>;\
<GB38—01>;…;<GB38—94>;\
<GB39—01>;…;<GB39—94>;\
<GB40—01>;…;<GB40—94>;\
<GB41—01>;…;<GB41—94>;\
<GB42—01>;…;<GB42—94>;\
<GB43—01>;…;<GB43—94>;\
<GB44—01>;…;<GB44—94>;\

<GB45—01>;…;<GB45—94>;\
<GB46—01>;…;<GB46—94>;\
<GB47—01>;…;<GB47—94>;\
<GB48—01>;…;<GB48—94>;\
<GB49—01>;…;<GB49—94>;\
<GB50—01>;…;<GB50—94>;\
<GB51—01>;…;<GB51—94>;\
<GB52—01>;…;<GB52—94>;\
<GB53—01>;…;<GB53—94>;\
<GB54—01>;…;<GB54—94>;\
<GB55—01>;…;<GB55—89>;\
<GB56—01>;…;<GB56—94>;\
<GB57—01>;…;<GB57—94>;\
<GB58—01>;…;<GB58—94>;\
<GB59—01>;…;<GB59—94>;\
<GB60—01>;…;<GB60—94>;\
<GB61—01>;…;<GB61—94>;\
<GB62—01>;…;<GB62—94>;\
<GB63—01>;…;<GB63—94>;\
<GB64—01>;…;<GB64—94>;\
<GB65—01>;…;<GB65—94>;\
<GB66—01>;…;<GB66—94>;\
<GB67—01>;…;<GB67—94>;\
<GB68—01>;…;<GB68—94>;\
<GB69—01>;…;<GB69—94>;\
<GB70—01>;…;<GB70—94>;\
<GB71—01>;…;<GB71—94>;\
<GB72—01>;…;<GB72—94>;\
<GB73—01>;…;<GB73—94>;\
<GB74—01>;…;<GB74—94>;\
<GB75—01>;…;<GB75—94>;\
<GB76—01>;…;<GB76—94>;\
<GB77—01>;…;<GB77—94>;\
<GB78—01>;…;<GB78—94>;\
<GB79—01>;…;<GB79—94>;\
<GB80—01>;…;<GB80—94>;\
<GB81—01>;…;<GB81—94>;\
<GB82—01>;…;<GB82—94>;\
<GB83—01>;…;<GB83—94>;\
<GB84—01>;…;<GB84—94>;\
<GB85—01>;…;<GB85—94>;\
<GB86—01>;…;<GB86—94>;\
<GB87—01>;…;<GB87—94>

在 6763 个中文字符中包含 186 个字根：

```
radical  <GB16-43>;<GB16—55>;<GB17—20>;<GB17—39>;<GB17—40>;\
         <GB18-23>;<GB18—41>;<GB19—07>;<GB19—21>;<GB19—28>;\
         <GB19-29>;<GB19—61>;<GB19—64>;<GB19—70>;<GB20—71>;\
         <GB20-83>;<GB20—85>;<GB21—22>;<GB22—23>;<GB22—25>;\
         <GB22—89>;<GB22—90>;<GB22—94>;<GB23—29>;<GB23—71>;\
         <GB24-24>;<GB24—74>;<GB24—79>;<GB25—04>;<GB25—13>;\
         <GB25—39>;<GB25—40>;<GB25—47>;<GB25—67>;<GB25—77>;\
         <GB26—44>;<GB26—58>;<GB27—07>;<GB27—80>;<GB28—24>;\
         <GB28-26>;<GB28—91>;<GB29—39>;<GB29—77>;<GB29—79>;\
         <GB29-80>;<GB30—42>;<GB31—58>;<GB32—47>;<GB32—79>;\
         <GB33-02>;<GB33—06>;<GB33—90>;<GB34—17>;<GB34—25>;\
         <GB34—73>;<GB34—77>;<GB34—83>;<GB35—11>;<GB35—12>;\
         <GB35-37>;<GB35—55>;<GB35—83>;<GB36—30>;<GB36—31>;\
         <GB36—81>;<GB37—03>;<GB37—14>;<GB38—04>;<GB38—12>;\
         <GB38—68>;<GB38—88>;<GB39—23>;<GB39—64>;<GB40—14>;\
         <GB40-43>;<GB40—53>;<GB41—29>;<GB41—64>;<GB41—77>;\
         <GB42-12>;<GB42—14>;<GB42—15>;<GB42—19>;<GB42—24>;\
         <GB42-30>;<GB42—31>;<GB42—54>;<GB42—83>;<GB43—14>;\
         <GB43-36>;<GB44—79>;<GB45—33>;<GB45—63>;<GB45—85>;\
         <GB46-04>;<GB46—36>;<GB46—67>;<GB46—87>;<GB47—06>;\
         <GB48-01>;<GB48—33>;<GB48—36>;<GB49—08>;<GB49—10>;\
         <GB49-52>;<GB49—82>;<GB50—19>;<GB50—21>;<GB50—27>;\
         <GB50-34>;<GB50—50>;<GB50—84>;<GB51—35>;<GB51—47>;\
         <GB51-54>;<GB51—67>;<GB51—74>;<GB51—80>;<GB52—34>;\
         <GB54-25>;<GB54—59>;<GB54—81>;<GB55—06>;<GB55—51>;\
         <GB55-52>;<GB55—63>;<GB55—67>;<GB56—13>;<GB56—15>;\
         <GB56-28>;<GB56—46>;<GB56—54>;<GB56—71>;<GB56—73>;\
         <GB57-72>;<GB57—79>;<GB57—91>;<GB58—02>;<GB58—05>;\
         <GB58-64>;<GB58—66>;<GB59—40>;<GB59—41>;<GB59—44>;\
         <GB60-19>;<GB62—35>;<GB62—44>;<GB62—48>;<GB63—14>;\
         <GB64-77>;<GB65—60>;<GB65—74>;<GB65—75>;<GB66—26>;\
         <GB66-27>;<GB66—64>;<GB67—60>;<GB67—63>;<GB69—18>;\
         <GB69-33>;<GB69—70>;<GB69—88>;<GB70—89>;<GB71—59>;\
         <GB71-61>;<GB74—23>;<GB75—22>;<GB76—15>;<GB76—65>;\
         <GB76-74>;<GB77—17>;<GB78—36>;<GB80—58>;<GB81—34>;\
         <GB81-66>;<GB81—71>;<GB82—14>;<GB83—30>;<GB84—62>;\
         <GB84-73>;<GB85—25>;<GB85—84>;<GB86—28>;<GB86—31>;\
         <GB87-52>

ENDLC_CTYPE
```

```
LC_COLLATE

# 在中国有各种不同的对中文进行分类排序的方法。比如下面的三种排序方法是经常用到的：
# 1) 按拼音排序；
# 2) 按偏旁部首和笔划多少排序；
# 3) 按笔划数排序。
# GB 2312 中文字符分两级，第一级(3755)和第二级(3008)。第一级按字符拼音排序，第二级则按偏旁
# 部首和笔划多少排序。


order_start forward

<NULL>
<SOH>
<STX>
<ETX>
<EOT>
<ENQ>
<ACK>
<alert>
<backspace>
<tab>
<newline>
<vertical-tab>
<form-feed>
<carriage-return>
<SO>
<SI>
<DLE>
<DC1>
<DC2>
<DC3>
<DC4>
<NAK>
<SYN>
<ETB>
<CAN>
<EM>
<SUB>
<ESC>
<IS4>
<IS3>
```

<IS2>
<IS1>
<space>
<exclamation—mark>
<quotation—mark>
<number—sign>
<dollar—sign>
<percent—sign>
<ampersand>
<apostrophe>
<left—parenthesis>
<right—parenthesis>
<asterisk>
<plus—sign>
<comma>
<hyphen>
<period>
<slash>
<zero>
<one>
<two>
<three>
<four>
<five>
<six>
<seven>
<eight>
<nine>
<colon>
<semicolon>
<less—than—sign>
<equals—sign>
<greater—than—sign>
<question—mark>
<commercial—at>
<A>
<B>
<C>
<D>
<E>
<F>
<G>
<H>

<I>
<J>
<K>
<L>
<M>
<N>
<O>
<P>
<Q>
<R>
<S>
<T>
<U>
<V>
<W>
<X>
<Y>
<Z>
<left—square—bracket>
<backslash>
<right—square—bracket>
<circumflex>
<underscore>
<grave—accent>
<a>
<b>
<c>
<d>
<e>
<f>
<g>
<h>
<i>
<j>
<k>
<l>
<m>
<n>
<o>
<p>
<q>
<r>
<s>

<t>
<u>
<v>
<w>
<x>
<y>
<z>
<left—brace>
<vertical—line>
<right—brace>
<tilde>
<DEL>

682 个 GB 2312 中的字符：
<GB01—01>
…
<GB01—94>
<GB02—17>
…
<GB02—66>
<GB02—69>
…
<GB02—78>
<GB02—81>
…
<GB02—92>
<GB03—01>
…
<GB03—94>
<GB04—01>
…
<GB04—83>
<GB05—01>
…
<GB05—86>
<GB06—01>
…
<GB06—24>
<GB06—33>
…
<GB06—56>
<GB07—01>
…

<GB07—33>
<GB07—49>
…
<GB07—81>
<GB08—01>
…
<GB08—26>
<GB08—37>
…
<GB08—73>
<GB09—04>
…
<GB09—79>

3755 个一级汉字：

<GB16—01>
…
<GB16—94>
<GB17—01>
…
<GB17—94>
<GB18—01>
…
<GB18—94>
<GB19—01>
…
<GB19—94>
<GB20—01>
…
<GB20—94>
<GB21—01>
…
<GB21—94>
<GB22—01>
…
<GB22—94>
<GB23—01>
…
<GB23—94>
<GB24—01>
…
<GB24—94>

<GB25—01>
…
<GB25—94>
<GB26—01>
…
<GB26—94>
<GB27—01>
…
<GB27—94>
<GB28—01>
…
<GB28—94>
<GB29—01>
…
<GB29—94>
<GB30—01>
…
<GB30—94>
<GB31—01>
…
<GB31—94>
<GB32—01>
…
<GB32—94>
<GB33—01>
…
<GB33—94>
<GB34—01>
…
<GB34—94>
<GB35—01>
…
<GB35—94>
<GB36—01>
…
<GB36—94>
<GB37—01>
…
<GB37—94>
<GB38—01>
…
<GB38—94>
<GB39—01>

…

<GB39—94>

<GB40—01>

…

<GB40—94>

<GB41—01>

…

<GB41—94>

<GB42—01>

…

<GB42—94>

<GB43—01>

…

<GB43—94>

<GB44—01>

…

<GB44—94>

<GB45—01>

…

<GB45—94>

<GB46—01>

…

<GB46—94>

<GB47—01>

…

<GB47—94>

<GB48—01>

…

<GB48—94>

<GB49—01>

…

<GB49—94>

<GB50—01>

…

<GB50—94>

<GB51—01>

…

<GB51—94>

<GB52—01>

…

<GB52—94>

<GB53—01>

…

<GB53—94>
<GB54—01>
…
<GB54—94>
<GB55—01>
…
<GB55—89>

3008 个二级汉字：

<GB56—01>
…
<GB56—94>
<GB57—01>
…
<GB57—94>
<GB58—01>
…
<GB58—94>
<GB59—01>
…
<GB59—94>
<GB60—01>
…
<GB60—94>
<GB61—01>
…
<GB61—94>
<GB62—01>
…
<GB62—94>
<GB63—01>
…
<GB63—94>
<GB64—01>
…
<GB64—94>
<GB65—01>
…
<GB65—94>
<GB66—01>
…
<GB66—94>

<GB67—01>
…
<GB67—94>
<GB68—01>
…
<GB68—94>
<GB69—01>
…
<GB69—94>
<GB70—01>
…
<GB70—94>
<GB71—01>
…
<GB71—94>
<GB72—01>
…
<GB72—94>
<GB73—01>
…
<GB73—94>
<GB74—01>
…
<GB74—94>
<GB75—01>
…
<GB75—94>
<GB76—01>
…
<GB76—94>
<GB77—01>
…
<GB77—94>
<GB78—01>
…
<GB78—94>
<GB79—01>
…
<GB79—94>
<GB80—01>
…
<GB80—94>
<GB81—01>

```
...
<GB81 -94>
<GB82 -01>
...
<GB82 -94>
<GB83 -01>
...
<GB83 -94>
<GB84 -01>
...
<GB84 -94>
<GB85 -01>
...
<GB85 -94>
<GB86 -01>
...
<GB86 -94>
<GB87 -01>
...
<GB87 -94>

order -- end

ENDLC _ COLLATE

LC _ MONETARY

# 货币符号<￥>,与汉字<圆>或<元>相等,它们的拼音为<yuan>。

int _ curr _ symbol          "<C><N><dollar-sign><space>"
currency _ symbol            "<C><N><GB03-04>"
mon _ decimal _ point        "<period>"
mon _ thousands _ sep        "<comma>"
mon _ grouping               3;0
positive _ sign              ""
negative _ sign              "<hyphen>"
int _ frac _ digits          2
frac _ digits                2
p _ cs _ precedes            1
p _ sep _ by _ space         0
n _ cs _ precedes            1
n _ sep _ by _ space         0
p _ sign _ posn              1
```

```
n_sign_posn                4

ENDLC_MONETARY

LC_NUMERIC

decimal_point          "<period>"
thousands_sep          "<comma>"
grouping               3;0

ENDLC_NUMERIC

LC_TIME

# 本章中的日期时间表示符合 GB/T 7408 标准。
# 用中文数字符号表示的<day>,<one>,<two>,<three>,<four>,<five>。
# <six>是星期几的缩略语。从星期日到星期六的缩略语是:
abday    "<GB40--53>";"<GB50—27>";"<GB22—94>";"<GB40—93>";\
         "<GB43—36>";"<GB46—69>";"<GB33—89>"

# 从星期日到星期六的全称是:

day      "<GB48—39><GB38—58><GB40—53>";\
         "<GB48—39><GB38—58><GB50—27>";\
         "<GB48—39><GB38—58><GB22—94>";\
         "<GB48—39><GB38—58><GB40—93>";\
         "<GB48—39><GB38—58><GB43—36>";\
         "<GB48—39><GB38—58><GB46—69>";\
         "<GB48—39><GB38—58><GB33—89>"

# 1 到 12 月份的缩略语:

abmon    "<one><GB52—34>";"<two><GB52—34>";\
         "<three><GB52—34>";"<four><GB52—34>";\
         "<five><GB52—34>";"<six><GB52--34>";\
         "<seven><GB52—34>";"<eight><GB52—34>";\
         "<nine><GB52—34>";"<one><zero><GB52—34>";\
         "<one><one><GB52--34>";"<one><two><GB52—34>"

# 1 到 12 月份的全称。
# 月份的数字部分用中文数字表示:

mon      "<GB50--27><GB52—34>";"<GB22—94><GB52—34>";\
```

```
        "<GB40—93><GB52—34>";"<GB43—36><GB52—34>";\
        "<GB46—69><GB52—34>";"<GB33—89><GB52—34>";\
        "<GB38—63><GB52—34>";"<GB16—43><GB52—34>";\
        "<GB30—37><GB52—34>";"<GB42—14><GB52—34>";\
        "<GB42—14><GB50—27><GB52—34>";\
        "<GB42—14><GB22—94><GB52—34>"

# 年用公元或公元前表示:
# era "+:0:0000/01/01:+*:<GB25—11><GB52—10>:%EC%Ey<GB36—74>";\
#     "+:1:-0001/12/31:-*:<GB25—11><GB52—10><GB39—16>:%EC%Ey<GB36—
      74>"
#
era   "<plus-sign><colon>\
<zero><colon>\
<zero><zero><zero><zero><    ><zero><one><slash><zero><one><colon>\
<plus-sign><asterisk><colon>\
<GB25—11><GB52—10><colon>\
<percent-sign><E><C><percent-sign><E><y><GB36—74>;\
      "<plus-sign><colon>\
<one><colon>\
<hyphen><zero><zero><zero><one><slash><one><two><slash><three>
<one><colon>\
<hyphen><asterisk><colon>\
<GB25—11><GB52—10><GB39—16><colon>\
<percent-sign><E><C><percent-sign><E><y><GB36—74>"

# 日期和时间的希望输出格式:
#
# e.g. 93YEAR 7 MONTH 10 DAY Thursday 12 HOUR 30 MINUTE 0 SECOND
#
# 其中的年月日时分秒用中文字符表示:
#
# d_t_fmt  "%E%m<GB52—34>%d<GB40—53>%<A>%H<GB42—17>%M<GB23—54
>%S
# <GB35—75>"
#
  d_t_fmt  "<percent-sign><E><percent-sign><m><GB52—34><percent-sign>
           <d>\
           <GB40—53><percent-sign><A><percent-sign><H><GB42—17><per-
           cent-sign>\
           <M><GB23—54><percent-sign><S><GB35—75>"
# %y/%m/%d(94/06/10)
#
```

```
# d—fmt   "%y<slash>%m<slash>%d"
d—fmt     "<percent—sign><y><slash><percent—sign><m><slash><percent—sign>
          <d>"
# %H:%M:%S(9:30:00)
#
# t—fmt   "%H<colon>%M<colon>%S"
t—fmt     "<percent—sign><H><colon><percent—sign><M><colon><percent—sign>
          <S>"
am—pm     "<GB41—47><GB46—71>";"<GB47—34><GB46—71>"

ENDLC _ TIME

LC _ MESSAGES
# yesexpr  "^ [<y><Y><GB03—57><GB03—89><GB42—39>]"
yesexpr    "<circumflex><left—square—bracket><y><Y><GB03—57>\
          <GB03—89><GB42—39>\
          <right—square—bracket>"
#  noexpr  "^ [<n><N><GB03—46><GB03—78><GB23—81>]"
noexpr     "<circumflex><left—square—bracket><n><N><GB03—78>\
          <GB03—46><GB23—81>\
          <right—square—bracket>"

END LC _ MESSAGES
```

附 录 B
（提示的附录）
实 现 考 虑

B1 通用输入方法子系统的结构与界面规格说明

B1.1 规格目标

好的输入方法子系统应具有以下特点：

a）国际性

可以根据系统中的 Locale 环境自动地装入或启动所需的输入方法模块，从而使系统及应用程序具有输入和处理多国语言的能力。

b）通用性

可以同时支持简、繁体中文输入，而且还可以进一步支持日、韩文等多字节文字输入，甚至可以提供对单字节文字输入的支持。

c）并发性

并发性有两个方面的含义：一是一个应用环境的输入方法子系统可以同时支持多个输入要求，如允许多个控制台屏幕或 X—Window 环境下的多个窗口上同时独立、互不干扰地使用同一个输入方法子系统进行不同的输入操作。

二是在一个系统中多个不同应用环境的输入方法子系统可以同时独立、互不干扰地运行。

d) 灵活性

用户可以很方便地增、删具体的输入法。具体的输入法开发者或用户可以在不需要深入了解输入方法子系统的结构及实现方法的情况下，很方便地加入自己开发的新输入法。

e) 友好性

支持多种风格的输入界面，同时允许用户根据自己的爱好修改或设计自己的输入界面。

f) 可移植性

对于根据本结构及规格实现的输入方法子系统及应用程序都可以很方便地移植到支持本结构的系统中。

B1.2 通用输入方法子系统的结构

通用输入方法子系统可按其功能分为：

a) 输入管理层

根据 Locale 环境自动到指定目录下找出所需的输入方法模块装入或启动应用程序，并填写有关入口表，以便建立应用程序与输入方法之间的联系。

b) 前端处理层对各个具体输入法进行管理，负责解释特殊键的含义，切换具体输入法，并把具体输入法处理的结果存放在相应的缓冲区中，以便应用程序取走或显示。

c) 输入单元层

根据输入单元的要求，对每个输入键事件进行解释，确定相应的动作，形成符合输入单元要求的输入串，并调用输入单元算法层的函数进行字典查找运算或代码转换运算，把查找结果返回前端处理层。

d) 输入单元算法层

根据上层函数送来的输入码串进行字典查找运算，并把查找结果返回上一层函数。

e) 辅助区处理层

提供不同风格的提示行、预编辑区、选字区的处理函数，以便向用户提供不同风格的输入界面。

其中，第 a、b、e 层构成了输入方法的应用程序编程界面(API)。第 e 层，即辅助区处理层应由应用程序实现，其界面应符合本规格要求。通用输入方法子系统各层间的关系如图 B1 所示。

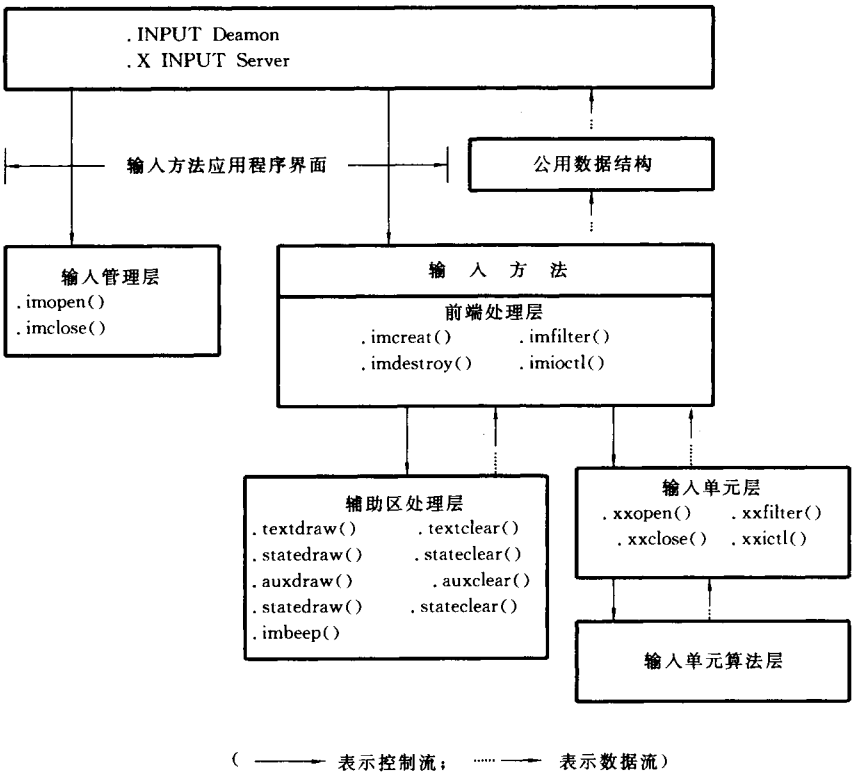

图 B1

附 录 C
(提示的附录)
参 考 文 献

XPG 3

X/Open Specification, 1988, 1989, February 1992, Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C21a); this specification was formerly X/Open Portability Guide, Volume 1, January 1989 XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/00b).

or

X/Open Specification, February 1992, System Interfaces and Headers. Issue 3 (ISBN: 1-872630-37-5, C21b); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interfaces and Headers (ISBN: 0-13-685843-0, XO/XPG/89/00c).

as appropriate

XPG 4

The XBD Specification, the XCU specification or the XSH specification as appropriate.

. System Interfaces Definitions, Issue 4 (the XBD Specification).

. Commands and Utilities, Issue 4 (the XCU Specification).

. System Interfaces and Headers, Issue 4 (the XSH Specification).

X Window System Protocal Version 11.

The X Font Service Protocal Version 1.0.

索　引

m　表示该界面被强制定义。

f　表示该界面是特征组的一部分，XPG4 文件中定义了这些界面是强制的还是可选的。

.　表示该界面没有被指定。

W　表示是世界通用界面(是强制的)。

续表

界　　面	章　条　号	XPG4	ISO/IEC 9945-1 和 9945-2	ISO/IEC 9899
isradical()	4.2.16　e)	.	.	.
isundefchar()	4.2.16　f)	.	.	.
iswalnum()	4.2.12　f)	W	.	.
iswalpha()	4.2.12　a)	W	.	.
iswcntrl()	4.2.12　k)	W	.	.
iswctype()	4.2.12　m)	W	.	.
iswdigit()	4.2.12　d)	W	.	.
iswgraph()	4.2.12　j)	W	.	.
iswlower()	4.2.12　c)	W	.	.
iswprint()	4.2.12　i)	W	.	.
iswpunct()	4.2.12　h)	W	.	.
iswspace()	4.2.12　g)	W	.	.
iswupper()	4.2.12　b)	W	.	.
iswxdigit()	4.2.12　e)	W	.	.
loaddispfont()	4.4.2.1.2　c)	.	.	.
localeconv()	4.2.5　a)	m	.	m
mblen()	4.2.1　b)	m	.	m
mbstowcs()	4.2.1　d)	m	.	m
mbtowc()	4.2.1　a)	m	.	.
nl_langinfo()	4.2.5　b)	m	.	.
PreeditCaretCallback()	4.6.2.4　r)	.	.	.
PreeditDoneCallback()	4.6.2.4　p)	.	.	.
PreeditDrawCallback()	4.6.2.4　q)	.	.	.
PreeditStartCallback()	4.6.2.4　o)	.	.	.
printf()	4.2.4　a)	m	m	m
putwchar()	4.2.2　g)	W	.	.
putwc()	4.2.2　f)	W	.	.
scanf()	4.2.10　a)	m	m	m
setdisplang()	4.4.2.1.2　a)	.	.	.
setlocale()	4.2.11　a)	m	m	m
sprintf()	4.2.4　a)	m	m	m
sscanf()	4.2.10　a)	m	m	m
stateclear()	4.4.1.1.4　d)	.	.	.
statedraw()	4.4.1.1.4　c)	.	.	.

续表

界　　面	章　条　号	XPG4	ISO/IEC 9945-1 和 9945-2	ISO/IEC 9899
StatusDrawCallback()	4.6.2.4　u)	.	.	.
StatusDoneCallback()	4.6.2.4　t)	.	.	.
StatusStartCallback()	4.6.2.4　s)	.	.	.
strcoll()	4.2.8　a)	m	.	m
strerror()	4.2.15　a)	m	.	m
strfmon()	4.2.5　c)	f	.	.
strftime()	4.2.6　a)	f	.	.
strptime()	4.2.6　b)	f	.	.
strtod()	4.2.9　a)	m	.	m
strtol()	4.2.9　b)	m	.	m
strtoul()	4.2.9　c)	m	.	m
strxfrm()	4.2.8　b)	m	.	m
textclear()	4.4.1.1.4　b)	.	.	.
textdraw()	4.4.1.1.4　a)	.	.	.
towlower()	4.2.13　b)	W	.	.
towupper()	4.2.13　a)	W	.	.
ungetwc()	4.2.2　i)	m	m	m
unloaddispfont()	4.4.2.1.2　d)	.	.	.
wcscat()	4.2.3　a)	W	.	.
wcschr()	4.2.3　h)	W	.	.
wcscmp()	4.2.3　c)	W	.	.
wcscoll()	4.2.8　c)	f	.	.
wcscpy()	4.2.3　e)	W	.	.
wcscspn()	4.2.3　l)	W	.	.
wcsftime()	4.2.6　c)	f	.	.
wcslen()	4.2.3　g)	W	.	.
wcsncat()	4.2.3　b)	W	.	.
wcsncmp()	4.2.3　d)	W	.	.
wcsncpy()	4.2.3　f)	W	.	.
wcspbrk()	4.2.3　j)	W	.	.
wcsrchr()	4.2.3　i)	W	.	.
wcsspn()	4.2.3　k)	W	.	.
wcstod()	4.2.9　d)	W	.	.
wcstok()	4.2.3　m)	W	.	.

续表

界　　面	章　条　号	XPG4	ISO/IEC 9945-1 和 9945-2	ISO/IEC 9899
wcstol()	4.2.9　e)	W	.	.
wcstombs()	4.2.1　e)	m	.	m
wcstoul()	4.2.9　f)	W	.	.
wcswcs()	4.2.3　n)	W	.	.
wcswidth()	4.2.14　b)	W	.	.
wcsxfrm()	4.2.8　d)	f	.	.
wctomb()	4.2.1　c)	.	.	.
wctype()	4.2.12　l)	W	.	.
wcwidth()	4.2.14　a)	W	.	.
XCloseIM()	4.6.2.4　b)	.	.	.
XCreateIC()	4.6.2.4　f)	.	.	.
XDestroyIC()	4.6.2.4　g)	.	.	.
XDisplayOfIM()	4.6.2.4　d)	.	.	.
XDrawScaledString()	4.6.1.1.2.2　b)	.	.	.
XDrawScaledWord()	4.6.1.1.2.2　a)	.	.	.
XFreeScaledFont()	4.6.1.1.2.1　d)	.	.	.
XGetICValues()	4.6.2.4　l)	.	.	.
XGetIMvalues()	4.6.2.4　c)	.	.	.
XIMOfIC()	4.6.2.4　k)	.	.	.
XLoadQueryScaledFont()	4.6.1.1.2.1　c)	.	.	.
XLoadScaledFont()	4.6.1.1.2.1　a)	.	.	.
XLocaleOfIM()	4.6.2.4　e)	.	.	.
XOpenIM()	4.6.2.4　a)	.	.	.
XQueryScaledFont()	4.6.1.1.2.1　b)	.	.	.
XSetICFocus()	4.6.2.4　h)	.	.	.
XSetICValues()	4.6.2.4　m)	.	.	.
XSetRotateAngle()	4.6.1.1.2.2　c)	.	.	.
XUnsetICFocus()	4.6.2.4　i)	.	.	.
XwcLookupString()	4.6.2.4　v)	.	.	.
XwcResetIC()	4.6.2.4　j)	.	.	.
xxclose()	4.4.1.1.3　b)	.	.	.
xxfilter()	4.4.1.1.3　c)	.	.	.
xxioctl()	4.4.1.1.3　d)	.	.	.
xxopen()	4.4.1.1.3　a)	.	.	.

中 华 人 民 共 和 国
国 家 标 准
信息技术 开放系统中文界面规范
GB/T 16681—1996
*
中国标准出版社出版
北京复兴门外三里河北街16号
邮政编码:100045
电 话:68522112
中国标准出版社秦皇岛印刷厂印刷
新华书店北京发行所发行 各地新华书店经售

*
开本 880×1230 1/16 印张 6½ 字数 200千字
1997年9月第一版 1998年6月第二次印刷
印数 501—1 000
*
书号:155066·1-14046 定价 46.00元
*
标 目 316—36

GB/T 16681-1996