CES Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

1 Library Overview 3
Purpose and Features 3
Main Functions 3
Consumed Resources 3
Embedding into Program 3
Sample Programs 3
Reference Materials 4
2 Feature Explanations and Usage Procedure 5
Unicode Character Encoding Scheme 5
Handling of the Unicode Character Encoding Scheme 6
Handling Unicode and Other CESs 10
3 Reference Information 16
User-Defined Characters for Japanese Handled in This Library 16

1 Library Overview

Purpose and Features

CES library is a library that supports handling of the Character Encoding Scheme (CES). By using the CES library, Unicode CES changes, interconversion with the CESs of each region, etc., can be done.

Main Functions

The following are the main functions offered by the CES library.

- Function to convert Unicode CES
  (UTF-32, UTF-32BE, UTF-32LE, UTF-16, UTF-16BE, UTF-16LE, UTF-8, UCS-2)
- Function to perform interconversion between Unicode and the CES of each region

Language	Overview of Corresponding CESs
European languages	ISO/IEC 8859-1 to 11, 13 to 16, ISO/IEC 646 Also, the code pages applied in the various corresponding regions.
Japanese	Shift_JIS, EUC-JP
Chinese	GB18030, GBK, EUC-CN (GB2312) , Big5
Korean	UHC, EUC-KR

Consumed Resources

The system resources used by the CES library are shown below.

Resource	Description
Footprint	The required part is linked to the executable file according to the function that is used. The size will be less than 400 KiB even if all the functions are used.

Embedding into Program

Include ces.h in the source program. (Additionally, a number of header files are automatically included.)

When building programs, link libSceCes.a.

Sample Programs

The following are provided as sample programs that use the CES library. Refer to these sample programs together with "2. Feature Explanations and Usage Procedure".

sample_code/system/api_libces/unicode/utf16txt_to_utf8txt

This sample converts text files described in UTF-16 to text files described in UTF-8 using the CES library and outputs the resulting files.

sample_code/system/api_libces/unicode/utf8_security

This sample program checks whether an invalid and unauthorized byte string for UTF-8 is not treated as a character when performing a function provided by the CES library to acquire UTF-32 (Unicode value) from UTF-8, and verifies that there is no factor which may lead up to software vulnerability.

sample_code/system/api_libces/multi_ces_converts

This sample program shows the method to implement conversion processing which handles the conversion of Unicode and multiple CESs by using the CES library.

sample_code/system/api_libces/string_class

This sample program shows the method to simply handle a character string and its encoding conversion by implementing each CES's character string class through the CES library.

Reference Materials

- Unicode
  Copyright © 1991-2011 Unicode, Inc. All rights reserved.
  The Unicode Consortium
  http://www.unicode.org

(The above reference destination has been confirmed as of November 18, 2014. Note that pages may have been subsequently moved or its contents modified.)

2 Feature Explanations and Usage Procedure

Unicode Character Encoding Scheme

Among the UCS (Unicode/ISO-10646) CESs, the CES library supports the following CESs.

CES	Size of one character	Expressible character code range	Reference
UTF-32, UTF-32BE, UTF-32LE	4 byte (32bit value)	U+00000000 to U+0010FFFF	
UTF-16, UTF-16BE, UTF-16LE	2byte(16bit value)	U+00000000 to U+0000FFFF	RFC 2781
	4 byte (two 16bit values)	U+00010000 to U+0010FFFF	
UTF-8	1 byte	U+00000000 to U+0000007F	RFC 3629
	2 byte	U+00000080 to U+000007FF	
	3 byte	U+00000800 to U+0000FFFF	
	4 byte	U+00010000 to U+0010FFFF	
UCS-2	2 byte (16bit value)	U+00000000 to U+0000FFFF	

UTF-32

This encoding scheme expresses Unicode characters with a fixed width of 32 bits per character.

The endian determination is possible by adding the character code U+FEFF called BOM (Byte Order Mark) to the beginning of the character string. Functions handling the UTF-32 character string support the BOM processing.

UTF-32BE and UTF-32LE

These indicate Big-endian UTF-32 and Little-endian UTF-32.

UTF-16

This scheme expresses each Unicode character with one or two 16-bit values.

Characters expressed with one 16-bit value are the same as UCS-2. The NULL character terminating a character string is also a 16-bit value. The characters between U+00010000 to U+00010FFFF are expressed with two 16-bit values.

The endian determination is possible by adding the character code U+FEFF called BOM (Byte Order Mark) to the beginning of the character string. Functions handling the UTF-16 character string support the BOM processing.

UTF-16BE and UTF-16LE

These indicate Big-endian UTF-16 and Little-endian UTF-16.

UTF-8

This scheme handles Unicode code points with a variable length byte stream.

One character can be expressed with 1 to 4 bytes. Characters expressed with the 1 byte from U+00 to U+7f are compatible with ASCII, and in the case of characters of multiple bytes, these characters consist of the value in the most significant bit.

UCS-2

This scheme handles Unicode code points with 16-bit scalar values.

It is not possible to express a character whose scalar value exceeds 16 bits. All expressible characters are expressed with a fixed size of 16 bits (2 bytes) per character.

Handling of the Unicode Character Encoding Scheme

The Unicode encoding schemes are expressed as follows in terms of the function names of the CES library.

`Utf32`, `Utf32be`, `Utf32le`, `Utf16`, `Utf16be`, `Utf16le`, `Utf8`, `Ucs2`

(When expressing a character string: `Utf32Str`, `Utf16Str`, `Utf8Str`, `Ucs2Str`)

UCS (Unicode/ISO-10646) is expressed as Ucs.

For each CES, the address types of a character and character string passed to a function is defined as follows.

CES	Type of the address of the character code and character string
UTF-32, UTF-32BE, UTF-32LE	`uint32_t*`
UTF-16, UTF-16BE, UTF-16LE	`uint16_t*` (variable-length)
UTF-8	`uint8_t*` (variable-length)
UCS-2	`uint16_t*`

A 1-character processing function handling both UTF-32 and UCS-2, which can express 1 character with a scalar value, uses the following character code type.

CES	Type of 1 character code value
UTF-32	`uint32_t`
UCS-2	`uint16_t`

For a function which handles a variable-length character code and character string, the maximum length of the buffer and the address of the variable receiving the character string length recognized by the function must be required as arguments as well as an address. The units of the required buffer length and the character string length to be returned are indicated with the unit number of the address type.

Example: 1-character code check

```
ret = sceCesUtf8CheckCode( &utf8code, utf8max, &utf8Len );
ret = sceCesUtf32beCheckCode( &utf32be );
ret = sceCesUtf32CheckCode( utf32code ); //Code value
```

Example: 1-character conversion

```
ret = sceCesUtf16ToUtf32( &utf16code, utf16max, &utf16Len, &utf32 );
ret = sceCesUtf32ToUtf16( utf32,
        &utf16buf[0], sizeof(utf16buf)/sizeof(uint16_t), &utf16Len );
```

The conversion function is named using the input CES name followed by the word [To] and the output CES name in that order. The order of arguments is the same as that of the function name; the arguments for input character (string) are put first and then for output character (string).

UTF-32 and UTF-16 are the CESs that the endian is determined with BOM. With regard to the 1-character processing function, however, the byte order is not considered because the BOM code is also treated as 1 character equally.

Functions handling a character string support the endian determination and endian designation using BOM, as well as the BOM output control.

Functions to check 1 Unicode character

```
sceCesUtf32CheckCode(), sceCesUtf32beCheckCode(), sceCesUtf32leCheckCode(),
sceCesUtf16CheckCode(), sceCesUtf16beCheckCode(), sceCesUtf16leCheckCode(),
sceCesUtf8CheckCode(), sceCesUcs2CheckCode()
```

Functions to convert CES of 1 Unicode character

```
sceCesUtf32ToUtf16(), sceCesUtf32ToUtf16be(), sceCesUtf32ToUtf16le(),
```

```
sceCesUtf32ToUtf8(), sceCesUtf32ToUcs2(),
sceCesUtf32beToUtf16(), sceCesUtf32beToUtf16be(), sceCesUtf32beToUtf16le(),
sceCesUtf32beToUtf8(), sceCesUtf32beToUcs2(),
sceCesUtf32leToUtf16(), sceCesUtf32leToUtf16be(), sceCesUtf32leToUtf16le(),
sceCesUtf32leToUtf8(), sceCesUtf32leToUcs2(),
sceCesUtf16ToUtf32(), sceCesUtf16ToUtf32be(), sceCesUtf16ToUtf32le(),
sceCesUtf16ToUtf8(), sceCesUtf16ToUcs2(),
sceCesUtf16beToUtf32(), sceCesUtf16beToUtf32be(), sceCesUtf16beToUtf32le(),
sceCesUtf16beToUtf8(), sceCesUtf16beToUcs2(),
sceCesUtf16leToUtf32(), sceCesUtf16leToUtf32be(), sceCesUtf16leToUtf32le(),
sceCesUtf16leToUtf8(), sceCesUtf16leToUcs2(),
sceCesUtf8ToUtf32(), sceCesUtf8ToUtf32be(), sceCesUtf8ToUtf32le(),
sceCesUtf8ToUtf16(), sceCesUtf8ToUtf16be(), sceCesUtf8ToUtf16le(),
sceCesUtf8ToUcs2(),
sceCesUcs2ToUtf32(), sceCesUcs2ToUtf32be(), sceCesUcs2ToUtf32le(),
sceCesUcs2ToUtf16(), sceCesUcs2ToUtf16be(), sceCesUcs2ToUtf16le(),
sceCesUcs2ToUtf8()
```

Functions handling a character string and the context

The name of a function handling a character string includes one of the following words which indicate the encoding scheme to be dealt with.

```
Utf32Str, Utf16Str, Utf8Str, Ucs2Str
```

For the first argument of the function, a context storing the settings and statuses relating to the character string conversion is required. Therefore, it is necessary to prepare an initialized context in advance. Providing an individual context for each thread enables functions handling a character string to be used in multithread processing.

As the functions handling a character string, the character string length acquisition function and the character string conversion and copy functions are provided in sets.

Since the following explanation only shows a basic usage procedure, refer to the reference document for details on the operation of the functions.

```
// Initializing a context
SceCesUcsContext ctx;
sceCesUcsContextInit( &ctx );

// Setting details on handling of BOM and endian to the context
sceCesSetUcsPolicyDetectBom( &ctx, SCE_CES_DETECT_ENABLE );
sceCesSetUcsPolicyOutputBom( &ctx, SCE_CES_OUTPUT_ENABLE );
sceCesSetUtf16StrEndian( &ctx, SCE_CES_ENDIAN_BE, SCE_CES_ENDIAN_SYS );
sceCesSetUtf32StrEndian( &ctx, SCE_CES_ENDIAN_STAY, SCE_CES_ENDIAN_SYS );

// Converting UTF-16 character strings to fixed-length UTF-32 character strings
ret = sceCesUtf16StrGetUtf32Len( &ctx, utf16str, utf16max, &utf16Len,
                                                          &utf32Len);
utf32max = utf32Len + 1; // +1: NULL termination character
utf32buf = malloc( sizeof(uint32_t) * utf32max );

ret = sceCesUtf16StrToUtf32Str( &ctx, utf16str, utf16max, &utf16Len,
                                      utf32buf, utf32max, &utf32Len);

// Coping UTF-8 character strings with BOM removed
sceCesSetUcsPolicyOutputBom( &ctx, SCE_CES_OUTPUT_TOP_CUT );

ret = sceCesUtf8StrGetCopyLen( &ctx, utf8bomStr, utf8bomMax, &utf8bomLen,
                                                             &utf8Len);
utf8max = utf8Len + 1; // +1:NULL termination character
utf8Str = malloc( sizeof(uint8_t) *  utf8max );
```

```
ret = sceCesUtf8StrToCopyStr( &ctx, utf8bomStr, utf8bomMax, &utf8bomLen,
                                    utf8Str   , utf8max   , &utf8Len);
```

Character string functions of this library have the following characteristics.

- With regard to the input character string, the part from the first character to the NULL termination character or up to which the character string length reaches the maximum buffer length is treated as a character string.
- If 0 is specified to the maximum buffer length of the input character string, it is assumed that there is no limitation for the length of the character string. Thus, the part from the first character to the NULL termination character is always recognized.
  In other words, the character string must be NULL terminated when 0 is specified, and it is not possible to pass the zero-sized buffer address.
- The buffer specified as the destination for storing the output character string is guaranteed to be NULL terminated in principle.
  Therefore, the size of the output buffer must include the size of the NULL termination character. If enough size of the output buffer is not allocated, an error will be returned.
- The maximum length of the buffer used for storing the output character string must be always specified. If the value is 0, an error indicating insufficient buffer size will be returned.
- The maximum length of the buffer specified to a character string function requires the address type's unit number of the character string to be encoded.
  In the case of UTF-16, the address type will be `uint16_t*`, and it is required to be specified in 16-bit word count.
- The length of the character string to be returned from a character string function is also the address type's unit number of the character string and does not include the size of the NULL termination character.

Functions for each CES described here is assumed to be used in the case where a `String` class is implemented for each CES or within processing for which a CES to be handled is determined.

When implementing a CES conversion processing, if it is necessary to implement multiple CES combinations, use integrated functions that can convert the specified CESs by internally calling the functions described here. For details, refer to "Handling a Variety of Unicode CES Conversions" and "Conversion of CESs between Unicode and other character sets through integrated functions".

Data type for handling Unicode character string

`SceCesUcsContext`

This data type holds the settings for a character string conversion of Unicode CES.

It is required to be specified for the first argument of a function which handles a character string.

Setting functions for a context which handles Unicode

```
sceCesSetUcsPolicyDetectBom(), sceCesSetUcsPolicyOutputBom(),
sceCesSetUtf32StrEndian(),sceCesSetUtf16StrEndian()
```

Character string length acquisition functions for each Unicode CES

```
sceCesUtf32StrGetUtf16Len(), sceCesUtf32StrGetUtf8Len(),
sceCesUtf32StrGetUcs2Len(),
sceCesUtf16StrGetUtf32Len(), sceCesUtf16StrGetUtf8Len(),
sceCesUtf16StrGetUcs2Len(),
sceCesUtf8StrGetUtf32Len(), sceCesUtf8StrGetUtf16Len(),
sceCesUtf8StrGetUcs2Len(),
sceCesUcs2StrGetUtf32Len(), sceCesUcs2StrGetUtf16Len(),
sceCesUcs2StrGetUtf8Len(),
sceCesUtf32StrGetCopyLen(), sceCesUtf16StrGetCopyLen(),
sceCesUtf8StrGetCopyLen(), sceCesUcs2StrGetCopyLen()
```

Functions to convert Unicode character string CESs and to copy the character strings

```
sceCesUtf32StrToUtf16Str(), sceCesUtf32StrToUtf8Str(),
sceCesUtf32StrToUcs2Str(),
sceCesUtf16StrToUtf32Str(), sceCesUtf16StrToUtf8Str(),
sceCesUtf16StrToUcs2Str(),
sceCesUtf8StrToUtf32Str(), sceCesUtf8StrToUtf16Str(),
sceCesUtf8StrToUcs2Str(),
sceCesUcs2StrToUtf32Str(), sceCesUcs2StrToUtf16Str(),
sceCesUcs2StrToUtf8Str(),
sceCesUtf32StrToCopyStr(), sceCesUtf16StrToCopyStr(),
sceCesUtf8StrToCopyStr(), sceCesUcs2StrToCopyStr()
```

Handling a Variety of Unicode CES Conversions

This library provides integrated functions which support all cases of Unicode CES conversion. The name of the functions includes the word "`UcsStr`" (hereinafter referred to as `UcsStr` function).

With regard to the argument structure of the `UcsStr` function, a value which indicates CES (for example, `SCE_CES_UTF8`) is specified for the argument indicating a character string before the address of the character string.

The address type of the character string is handled with `void*` regardless of CES, and the unit of the maximum buffer length and the size and length of the character string returned from the function is byte.

```
// Initializing a context
SceCesUcsContext ctx;
sceCesUcsContextInit( &ctx );

// Getting the character string size required for the specified encoding scheme
expression with the size of NULL termination character included
ret = sceCesUcsStrGetEncodingSize( &ctx,
                                   SCE_CES_UTF16LE, srcStr, srcMax, &srcLen,
                                   SCE_CES_UTF16BE, &dstSize );
dstMax = dstSize;
dstBuf = malloc( dstMax );

// Outputting the converted character string into the allocated buffer
ret = sceCesUcsStrConvertEncoding( &ctx,
                                    SCE_CES_UTF16LE, srcStr, srcMax, &srcLen,
                                    SCE_CES_UTF16BE, dstBuf, dstMax, &dstLen );
```

Note that the character string length which does not include the size of the NULL termination character (byte count) will be returned to the arguments whose variable's name includes '`Len`' in the above example.

For example, specifying `SCE_CES_UTF32` for the output of `sceCesUcsStrConvertEncoding()` means that another 4 bytes are further added as the NULL termination character.

`UcsStr` functions are useful when it is necessary to create a multi-CES converter since the functions can handle Unicode and other encoding scheme. Details on this feature are explained in the "Handling Unicode and Other CESs" section subsequently.

Handling Unicode and Other CESs

In this library, conversion tables, feature resources, etc. which are required to perform interconversion between Unicode and other CESs are managed in a profile form (hereinafter referred to as UCS conversion profile).

Although the UCS conversion profile is provided for each CES, the profiles are roughly classified into the following two groups.

- UCS conversion profile for single-byte character set
- UCS conversion profile for multi-byte character set

The address of the UCS conversion profile is required as the argument for functions to convert 1 Unicode character and 1 character other than Unicode or functions to initialize a context required for character string conversion.

UCS conversion Profile for Single-Byte Character Set

UCS conversion profile for single-byte character set (SBCS) can be referred by using the return value of the reference functions whose name is beginning with the word "`sceCesRefersUcsProfile`".

Conversion tables, resources, etc. to be linked to a program are determined according to the reference function call as described in the following table.

7bit Single-Byte Character Set

API	ANSI, JIS, GB, KS
`sceCesRefersUcsProfileAscii()`	`ASCII`
`sceCesRefersUcsProfileJisX0201Roman()`	`JIS X 0201 Roman`
`sceCesRefersUcsProfileJisX0201RomanTilde0x7e()`	`JIS X 0201 Roman (0x7e:Tilde)`
`sceCesRefersUcsProfileGbT1988()`	`GB/T 1988`
`sceCesRefersUcsProfileGbT1988Tilde0x7e()`	`GB/T 1988(0x7e:Tilde)`
`sceCesRefersUcsProfileKsX1003()`	`KS X 1003`
`sceCesRefersUcsProfileKsX1003Tilde0x7e()`	`KS X 1003(0x7e:Tilde)`

8bit Single-Byte Character Set

API	ISO/IEC 8859
`sceCesRefersUcsProfileIso8859_1()`	`ISO/IEC 8859-1:1998`
`sceCesRefersUcsProfileIso8859_2()`	`ISO/IEC 8859-2:1999`
`sceCesRefersUcsProfileIso8859_3()`	`ISO/IEC 8859-3:1999`
`sceCesRefersUcsProfileIso8859_4()`	`ISO/IEC 8859-4:1998`
`sceCesRefersUcsProfileIso8859_5()`	`ISO/IEC 8859-5:1999`
`sceCesRefersUcsProfileIso8859_6()`	`ISO/IEC 8859-6:1999`
`sceCesRefersUcsProfileIso8859_7()`	`ISO/IEC 8859-7:2003`
`sceCesRefersUcsProfileIso8859_8()`	`ISO/IEC 8859-8:1999`
`sceCesRefersUcsProfileIso8859_9()`	`ISO/IEC 8859-9:1999`
`sceCesRefersUcsProfileIso8859_10()`	`ISO/IEC 8859-10:1998`
`sceCesRefersUcsProfileIso8859_11()`	`ISO/IEC 8859-11:2001`
`sceCesRefersUcsProfileIso8859_13()`	`ISO/IEC 8859-13:1998`
`sceCesRefersUcsProfileIso8859_14()`	`ISO/IEC 8859-14:1998`
`sceCesRefersUcsProfileIso8859_15()`	`ISO/IEC 8859-15:1999`
`sceCesRefersUcsProfileIso8859_16()`	`ISO/IEC 8859-16:2001`
API	**RFC, JIS**
`sceCesRefersUcsProfileKoi8R()`	`KOI8-R`
`sceCesRefersUcsProfileKoi8U()`	`KOI8-U`
`sceCesRefersUcsProfileJisX0201()`	`JIS X 0201`
`sceCesRefersUcsProfileJisX0201Tilde0x7e()`	`JIS X 0201(0x7e:Tilde)`

Single-Byte Code Page Character Set

API	Microsoft Windows Code Page
sceCesRefersUcsProfileCp1250()	Code Page 1250
sceCesRefersUcsProfileCp1251()	Code Page 1251
sceCesRefersUcsProfileCp1252()	Code Page 1252
sceCesRefersUcsProfileCp1253()	Code Page 1253
sceCesRefersUcsProfileCp1254()	Code Page 1254
sceCesRefersUcsProfileCp1255()	Code Page 1255
sceCesRefersUcsProfileCp1256()	Code Page 1256
sceCesRefersUcsProfileCp1257()	Code Page 1257
sceCesRefersUcsProfileCp1258()	Code Page 1258
API	**OEM Code Page**
sceCesRefersUcsProfileCp437()	Code Page 437
sceCesRefersUcsProfileCp737()	Code Page 737
sceCesRefersUcsProfileCp775()	Code Page 775
sceCesRefersUcsProfileCp850()	Code Page 850
sceCesRefersUcsProfileCp852()	Code Page 852
sceCesRefersUcsProfileCp855()	Code Page 855
sceCesRefersUcsProfileCp857()	Code Page 857
sceCesRefersUcsProfileCp858()	Code Page 858
sceCesRefersUcsProfileCp860()	Code Page 860
sceCesRefersUcsProfileCp861()	Code Page 861
sceCesRefersUcsProfileCp862()	Code Page 862
sceCesRefersUcsProfileCp863()	Code Page 863
sceCesRefersUcsProfileCp864()	Code Page 864
sceCesRefersUcsProfileCp865()	Code Page 865
sceCesRefersUcsProfileCp866()	Code Page 866
sceCesRefersUcsProfileCp869()	Code Page 869
sceCesRefersUcsProfileCp874()	Code Page 874

The type of return value of functions referring to the UCS conversion profile is `const SceCesSbcsUcsProfile*`.

By using this type, it is possible to use functions exclusively provided for single-byte character set.

The 1-character conversion functions having a simple argument structure to handle the single-byte character code with `uint8_t` type are provided.

Example: 1 character conversion

```
const SceCesSbcsUcsProfile* iso8859_15 = sceCesRefersUcsProfileIso8859_15();
uint8_t sbc = 0xA4;

ret = sceCesSbcToUtf32( iso8859_1, sbc, &utf32code );
ret = sceCesUtf32ToSbc( utf32code, iso8859_1, &sbc );
```

Character string functions are not provided for single-byte character set. Instead, by using `sceCesGetMbcsUcsProfile()` macro function, the `SceCesSbcsUcsProfile` type can be referred through UCS conversion profile type of multi-byte character set (described below). Thus, it becomes possible to handle character strings through the multi-byte character string functions.

For details on the specific usage method, refer to the explanation of UCS conversion profile for multi-byte character set.

Type of UCS conversion profile for single-byte character set (SBCS)

`SceCesSbcsUcsProfile`

Functions to convert CESs of single-byte character set (SBCS) characters and Unicode characters

```
sceCesSbcToUtf32(), sceCesSbcToUtf32be(), sceCesSbcToUtf32le(),
sceCesSbcToUtf16(), sceCesSbcToUtf16be(), sceCesSbcToUtf16le(),
```

```
sceCesSbcToUtf8(), sceCesSbcToUcs2()
sceCesUtf32ToSbc(), sceCesUtf32beToSbc(), sceCesUtf32leToSbc()
sceCesUtf16ToSbc(), sceCesUtf16beToSbc(), sceCesUtf16leToSbc()
sceCesUtf8ToSbc(), sceCesUcs2ToSbc()
```

UCS conversion profile for multi-byte character set

The UCS conversion profile for multi-byte character set can be acquired through initialization functions constituting the UCS conversion profile.

An initialization function is provided for each multi-byte character set so as to select the function depending on the intended use.

Conversion tables, resources, etc. to be linked to a program are determined according to the initialization function call as described in the following table.

Shift JIS and UCS conversion profile initialization functions

API	Description
`sceCesUcsProfileInitSJis()` `(sceCesUcsProfileInitSJis1997Cp932())`	ASCII + JIS X 0208 + User-defined character set for Japanese (Microsoft Code Page 932 based character set and mapping)
`sceCesUcsProfileInitSJis1997X0208()`	JIS X 0201 + JIS X 0208(Shift_JIS)
`sceCesUcsProfileInitSJis2004X0213()`	ASCII + JIS X 0201 Katakana+JIS X 0213:2004 (Shift_JIS2004)

EUC-JP and UCS conversion profile initialization functions

API	Description
`sceCesUcsProfileInitEucJp()` `(sceCesUcsProfileInitEucJpCp51932())`	ASCII + JIS X 0208 + JIS X 0201 Katakana + User-defined character set for Japanese
`sceCesUcsProfileInitEucJpX0208()`	ASCII + JIS X 0208
`sceCesUcsProfileInitEucJpX0208Ss2()`	ASCII + JIS X 0208 + JIS X 0201 Katakana
`sceCesUcsProfileInitEucJpX0208Ss2Ss3()`	ASCII + JIS X 0208 + JIS X 0201 Katakana + JIS X 0212
`sceCesUcsProfileInitEucJis2004()`	ASCII + JIS X 0213:2004 + JIS X 0201 Katakana + JIS X 0212

Big5 and UCS conversion profile initialization functions

API	Description
`sceCesUcsProfileInitBig5()` `(sceCesUcsProfileInitBig5Cp950())`	ASCII + Big5 (Microsoft Code Page 950 based character set and mapping)

GB and UCS conversion profile initialization functions

API	Description
`sceCesUcsProfileInitGbk()` `(sceCesUcsProfileInitGbkCp936())`	GBK(Microsoft Code Page 950 based character set and mapping)
`sceCesUcsProfileInitGb18030()` `(sceCesUcsProfileInitGb18030_2000())`	ASCII + GB18030:2000
`sceCesUcsProfileInitGb18030_2005()`	ASCII + GB18030:2005

EUC-CN and UCS conversion profile initialization functions

API	Description
`sceCesUcsProfileInitEucCn()` `(sceCesUcsProfileInitEucCnGb2312())`	ASCII + GB2312

UHC and UCS conversion profile initialization functions

API	Description
`sceCesUcsProfileInitUhc()`	ASCII + UHC (Unified Hangle Code:Microsoft Code Page 949 character set and mapping)

EUC-KR and UCS conversion profile initialization functions

API	Description
sceCesUcsProfileInitEucKr()	ASCII + KS X 1001:2002

For performing initialization, users have to prepare SceCesUcsProfileSheet type as an entity of the profile.

```
static SceCesUcsProfileSheet ProfSheet[1];
SceCesSJisUcsProfile* sjis = sceCesUcsProfileInitSJis( &ProfSheet[0] );
const SceCesMbcsUcsProfile* prof;

prof = sceCesGetMbcsUcsProfile(sjis);

// 1-character conversion
ret = sceCesUtf32ToMbc( utf32code, prof, mbcBuf, mbcBufMax, &mbcLen );

// Character string conversion
SceCesMbcsUcsContext ctx;
sceCesMbcsUcsContextInit( &ctx, prof );
sceCesSetUtf16StrEndian( &ctx, SCE_CES_ENDIAN_STAY, SCE_CES_ENDIAN_LE );

ret = sceCesMbcsStrGetUtf16Len( &ctx, mbcsStr, mbcsMax, &mbcsLen,
                                                        &utf16Len );
utf16max = utf16Len + 1;
utf16buf = malloc( sizeof(uint16_t)*utf16max );

ret = sceCesMbcsStrToUtf16Str( &ctx, mbcsStr, mbcsMax, &mbcsLen,
                                    utf16buf, utf16max, &utf16Len );
```

Types to handle Unicode and other encoded character string

SceCesUcsProfileSheet

This data type makes up the conversion information between Unicode and other character sets.

Users have to prepare this data type. The address of the following UCS conversion profile types can be acquired by using an initialization function which conctitutes the conversion information between Unicode and each multi-byte character set.

```
SceCesSJisUcsProfile, SceCesEucJpUcsProfile,
SceCesBig5UcsProfile, SceCesGbUcsProfile, SceCesEucCnUcsProfile,
SceCesUhcUcsProfile, SceCesEucKrUcsProfile
```

SceCesMbcsUcsProfile

This data type holds the conversion information between Unicode and multi-byte character sets.

The reference address of this data type can be acquired from the address of the UCS conversion profile of each CES through a macro function, sceCesGetMbcsUcsProfile().

SceCesMbcsUcsContext

This is a data type of the context used for character string conversion between Unicode and other character sets.

Specify SceCesMbcsUcsProfile for sceCesMbcsUcsContextInit() to perform initialization.

Functions to convert 1-character of multi-byte character and Unicode character

```
sceCesSbcToUtf32(), sceCesSbcToUtf32be(), sceCesSbcToUtf32le(),
sceCesSbcToUtf16(), sceCesSbcToUtf16be(), sceCesSbcToUtf16le(),
sceCesSbcToUtf8(), sceCesSbcToUcs2()
sceCesUtf32ToSbc(), sceCesUtf32beToSbc(), sceCesUtf32leToSbc()
```

```
sceCesUtf16ToSbc(), sceCesUtf16beToSbc(), sceCesUtf16leToSbc()
sceCesUtf8ToSbc(), sceCesUcs2ToSbc()
```

Functions to acquire the converted character string length of multi-byte character set and Unicode

```
sceCesMbcsStrGetUtf32Len(), sceCesUtf32StrGetMbcsLen(),
sceCesMbcsStrGetUtf16Len(), sceCesUtf16StrGetMbcsLen(),
sceCesMbcsStrGetUtf8Len(), sceCesUtf8StrGetMbcsLen(),
sceCesMbcsStrGetUcs2Len(), sceCesUcs2StrGetMbcsLen()
```

Functions to convert character string of multi-byte character set and Unicode

```
sceCesMbcsStrToUtf32Str(), sceCesUtf32StrToMbcsStr(),
sceCesMbcsStrToUtf16Str(), sceCesUtf16StrToMbcsStr(),
sceCesMbcsStrToUtf8Str(), sceCesUtf8StrToMbcsStr(),
sceCesMbcsStrToUcs2Str(), sceCesUcs2StrToMbcsStr()
```

Conversion of CESs between Unicode and other character sets through integrated functions

This library provides integrated conversion functions which can collectively support the cases of conversion between Unicode and other CESs.

These functions include the word "UcsStr" in their name as already described in "Handling a Variety of Unicode CES Conversions" (hereinafter referred to as UcsStr function). With regard to the argument structure of the UcsStr function, a value which indicates CES (for example, SCE_CES_UTF8) is specified for the argument indicating a character string before the argument of the character string's address. The address type of the character string is handled with void* regardless of CES, and the unit of the maximum buffer length and the length and size of the character string returned from the function is byte.

Basically, the UcsStr functions only support Unicode character strings. However, using a context initialized with sceCesMbcsUcsContextInit() allows the user to specify SCE_CES_MBCS for the value indicating the CES.

Consequently, it is possible to support the character string conversion to handle Unicode CES and an arbitrary CES indicated by the context. (Note that the UcsStr functions are the functions that handle Unicode character strings and therefore, SCE_CES_MBCS cannot be specified for both input and output.)

```
// Preparing the UCS conversion profile
SceCesUcsProfileSheet ProfSheet[1];
SceCesSJisUcsProfile*       sjis;
const SceCesSbcsUcsProfile* iso8859_15;

sjis = sceCesUcsProfileInitSJis( &ProfSheet[0] );
iso8859_15 = sceCesRefersUcsProfileIso8859_15();


// Performing conversion from MBCS character string to UTF-16 character string
const SceCesMbcsUcsProfile* mProf = (SceCesMbcsUcsProfile*)0;
SceCesMbcsUcsContext mCtx;
SceCesUcsContext* ctx;

switch( mCesType ) {
case SJIS      : mProf = sceCesGetMbcsUcsProfile( sjis );    break;
     :
case ISO8859_15: mProf = sceCesGetMbcsUcsProfile( iso8859 ); break;
default        : break;
}

sceCesMbcsUcsContextInit( &mCtx, mProf );
ctx = sceCesGetUcsContext( &mCtx );  //&mCtx->cesUcsCtx;

ret = sceCesUcsStrGetEncodingSize( &ctx,
```

```
                                        SCE_CES_MBCS, srcStr, srcMax, &srcLen,
                                        SCE_CES_UTF16, &dstSize );
    dstMax = dstSize;
    dstBuf = malloc( dstMax );
    ret = sceCesUcsStrConvertEncoding( &ctx,
                                        SCE_CES_MBCS, srcStr, srcMax, &srcLen,
                                        SCE_CES_UTF16, dstBuf, dstMax, &dstLen );
```

In the above example, the character string length which does not include the size of the NULL termination character (byte count) is returned to the arguments whose variable's name includes 'Len'. For example, specifying `SCE_CES_UTF32` for the output of `sceCesUcsStrConvertEncoding()` means that another 4 bytes are further written as the NULL termination character.

3 Reference Information

User-Defined Characters for Japanese Handled in This Library

Table 1 The Code Range of User-Defined Character for Japanese

Type of User-Defined Character for Japanese	Row (Shift JIS)	Shift JIS Code Value	EUC-JP Code Value
NEC special characters	13th row	0x8740 to 0x879c	0xADA1 to 0xADFC
NEC-selected IBM extended characters	89th to 92nd row	0xed40 to 0xeefc	0xF9A1 to 0xFCFE
IBM extended characters	115th to 119th row	0xfa40 to 0xfc4b	