XC Library

IN THIS DOCUMENT

The XC library provides a set of supporting functions and typedefs that make it easier to write code that can be called from C, C++ and XC.

1 print.h functions

The `print` library provides support for formatted output.

`int printchar(char value)`

Prints a character.

This function has the following parameters:

`value`	The character to print.

This function returns:

The number of characters printed, or -1 on error.

`int printcharln(char value)`

Prints a character followed by a new line.

This function has the following parameters:

`value`	The character to print.

This function returns:

The number of characters printed, or -1 on error.

`int printint(int value)`

Prints a value as a signed decimal.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printintln(int value)`

Prints a value as a signed decimal followed by a newline.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printuint(unsigned value)`

Prints a value as a unsigned decimal.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printuintln(unsigned value)`

Prints a value as a unsigned decimal followed by a newline.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printllong(long long value)`

Prints a long long value as a signed decimal.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printllongln(long long value)`

Prints a long long value as a signed decimal followed by a newline.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printullong(unsigned long long value)`

Prints a long long value as a unsigned decimal.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printullongln(unsigned long long value)`

Prints a long long value as a unsigned decimal followed by a newline.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printhex(unsigned value)`

Prints a value as a unsigned hexadecimal.

The upper-case letters are used for the conversion.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printhexln(unsigned value)`

Prints a value as a unsigned hexadecimal followed by a newline.

The upper-case letters are used for the conversion.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printllonghex(unsigned long long value)`

Prints a long long value as a unsigned hexadecimal.

The upper-case letters are used for the conversion.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printllonghexln(unsigned long long value)`

Prints a long long value as a unsigned hexadecimal followed by a newline.

The upper-case letters are used for the conversion.

This function has the following parameters:

`value`	The value to print.

This function returns:

The number of characters printed, or -1 on error.

`int printstr(const char s[])`

Prints a null terminated string.

This function has the following parameters:

`s`	The string to print.

This function returns:

The number of characters printed, or -1 on error.

`int printstrln(const char s[])`

Prints a null terminated string followed by a newline.

This function has the following parameters:

`s`	The string to print.

This function returns:

The number of characters printed, or -1 on error.

2 safestring.h functions

The `safestring` library provides safe versions of the string functions found in `string.h` of the C standard library. All functions are callable from XC. When called from XC any attempt to perform an out of bounds array access will cause an exception to be raised.

`void safestrcpy(char s1[], const char s2[])`

Copies a string (including the terminating null character) to an array.

This function has the following parameters:

`s1`	The array to copy to.
`s2`	The string to copy.

`void safestrncpy(char s1[], const char s2[], unsigned n)`

Copies no more than `n` characters of the string `s1` to the array `s2`.

If the length of `s2` is less than `n` then null characters will be appended to the copied characters until `n` bytes are written.

This function has the following parameters:

`s1`	The array to copy to.
`s2`	The string to copy.
`n`	The number of characters to copy.

`void safestrcat(char s1[], const char s2[])`

Appends a copy of a string (including the terminating null character) to the end of another string.

This function has the following parameters:

`s1`	The string to append to.
`s2`	The string to append.

`void safestrncat(char s1[], const char s2[], unsigned n)`

Appends no more than `n` characters of the string `s2` to the string `s1`.

The null characters at the end of `s1` is overwritten by the first character of `s2`. A terminating null character is always appended to the result.

This function has the following parameters:

`s1`	The string to append to.
`s2`	The string to append.
`n`	The number of characters to append.

`int safestrcmp(const char s1[], const char s2[])`

Compares two strings.

If the strings are equal 0 is returned. If the strings are not equal a non-zero value is returned, the sign of which is determined by the sign of the difference between the first pair of characters which differ in the strings being compared.

This function has the following parameters:

`s1`	The first string to compare.
`s2`	The second string to compare.

This function returns:

A integer greater than, equal to, or less than 0, if `s1` is respectively greater than, equal to, or less than `s2`.

`int safestrncmp(const char s1[], const char s2[], unsigned n)`

Compares up to the first `n` character of two strings.

If the strings are equal up to the first `n` characters, 0 is returned. Otherwise a non-zero value is returned, the sign of which is determined by the sign of the difference between the first pair of characters which differ.

This function has the following parameters:

`s1`	The first string to compare.
`s2`	The second string to compare.
`n`	The maximum number of characters to compare.

This function returns:

A integer greater than, equal to, or less than 0, if `s1` is respectively greater than, equal to, or less than `s2`.

`int safestrlen(const char s[])`

Returns the length of a string.

The length is given by the number of characters in the string not including the terminating null character.

This function has the following parameters:

`s`	The string.

This function returns:

The length of the string.

`int safestrchr(const char s[], int c)`

Returns the index of the first occurrence of `c` (converted to a `char`) in `s`.

If `c` does not occur in `s`, -1 is returned. The terminating null character is considered to be part of `s`.

This function has the following parameters:

`s`	The string to scan.
`c`	The character to scan for.

This function returns:

The index of `c`.

`int safestrrchr(const char s[], int c)`

Returns the index of the last occurrence of `c` (converted to a `char`) in `s`, or -1 if `c` does not occur in `s`.

The terminating null character is considered to be part of `s`.

This function has the following parameters:

`s`	The string to scan.
`c`	The character to scan for.

This function returns:

The index of `c`.

`unsigned safestrspn(const char s1[], const char s2[])`

Returns the length of the longest initial segment of `s1` which consists entirely of characters from `s2`.

This function has the following parameters:

`s1`	The string to scan.
`s2`	The string containing characters to scan for.

This function returns:

The length of the initial segment.

`unsigned safestrcspn(const char s1[], const char s2[])`

Returns the length of the longest initial segment of `s1` which consists entirely of characters not from `s2`.

This function has the following parameters:

`s1`	The string to scan.

`s2`	The string containing characters to scan for.

This function returns:

The length of the initial segment.

```
int safestrpbrk(const char s1[], const char s2[])
```

Returns the index of the first occurrence in `s1` of any character in `s2`.

If no character in `s2` occurs in `s1`, -1 is returned.

This function has the following parameters:

`s1`	The string to scan.
`s2`	The string containing characters to scan for.

This function returns:

The index of first matching character.

```
int safestrstr(const char s1[], const char s2[])
```

Returns the index of the first occurrence of `s1` as a sequence of characters (excluding the terminating null character) in `s2`.

If `s1` is not contained in `s2`, -1 is returned. If `s2` is a zero length string then 0 is returned.

This function has the following parameters:

`s1`	The string to scan.
`s2`	The string to scan for.

This function returns:

The index of first matching subsequence.

```
void safememcpy(unsigned char dst[],
                const unsigned char src[],
                unsigned length)
```

Copies `length` bytes from the array `src` to the array `dst`.

This function has the following parameters:

`dst`	The destination array.
`src`	The source array.
`length`	The number of bytes to copy.

```
void safememmove(unsigned char data[],
                 unsigned dst,
                 unsigned src,
                 unsigned length)
```

Copies `length` bytes from offset `src` of array `data` to offset `dst` of array `data`.

If the source and destination areas overlap then copying takes place as if the bytes are first copied from the source into a temporary array and then copied to the destination.

This function has the following parameters:

`data`	The array to move data in.
`dst`	The offset of the destination area.
`src`	The offset of the source area.
`length`	The number of bytes to copy.

```
void safememset(unsigned char dst[], int value, unsigned length)
```

Copies `value` (converted to an `unsigned char`) to each of the first `length` bytes of the array `dst`.

This function has the following parameters:

`dst`	The destination array.
`value`	The value to copy.
`length`	The number of bytes to copy.

```
int safememcmp(const unsigned char s1[],
               const unsigned char s2[],
               unsigned length)
```

Compares the first `length` bytes of the arrays `s1` and `s2`.

If there is no difference 0 is returned, otherwise a non-zero value is returned, the sign of which is determined by the sign of the difference between the first pair of bytes which differ.

This function has the following parameters:

`s1`	The first array.
`s2`	The second array.
`length`	The number of bytes to compare.

This function returns:

A integer greater than, equal to, or less than 0, if the first `length` bytes of `s1` are respectively greater than, equal to, or less than the first `length` bytes of `s2`.

`int safememchr(const unsigned char s[], int c, unsigned length)`

Returns the index of the first occurrence of `c` (converted to an `unsigned char`) in the first `length` bytes of `s`.

If `c` does not occur in `s`, -1 is returned.

This function has the following parameters:

`s`	The array to scan.
`c`	The character to scan for.
`length`	The number of bytes to scan.

This function returns:

The index of `c`.

3 xccompat.h typedefs

The `xccommpat` header file provides type definitions that simplify the task of writing functions that may be called from both C/C++ and XC.

`chanend`

chanend typedef for use in C/C++ code.

This typedef is only supplied if xccompat.h is included from C/C++ code. This enables a XC function prototyped as taking a parameter of type chanend to be called from C and vice versa.

`timer`

timer typedef for use in C/C++ code.

This typedef is only supplied if xccompat.h is included from C/C++ code. This enables a XC function prototyped as taking a parameter of type timer to be called from C and vice versa.

`port`

port typedef for use in C/C++ code.

This typedef is only supplied if xccompat.h is included from C/C++ code. This enables a XC function prototyped as taking a parameter of type port to be called from C and vice versa.