XBasic

Program Development Environment

(PDE)

Function Libraries

Math Library

Complex Number Library

Standard Library

Revision 0.0020

February 1, 1996

Table of Contents

Function Libraries Important Notes

Information in* `.dec` *files is always up to date.

In case of confusion or contradiction with this document, the* `.dec` *files are more reliable.

The* `.dec` *files may contain library functions not yet added to this document.

`IMPORT "xst" -` ***Standard Library***
`IMPORT "xui" -` ***GuiDesigner Library***
`IMPORT "xgr" -` ***GraphicsDesigner Library***
`IMPORT "xma" -` ***Advanced Math Library***
`IMPORT "xcm" -` ***Complex Math Library***

Math Function Library

Introduction

The *math function library* contains a comprehensive set of mathematics functions, including:

- ***trigonometric***
- ***arc-trigonometric***
- ***hyperbolic***
- ***arc-hyperbolic***
- ***logarithmic (base e and base 10)***
- ***exponential (base e and base 10)***
- ***miscellaneous (square root, power, etc.)***

Function Names

The names of library functions usually begin with a three character prefix that identifies the library - in this case it would be **`Xma`**.

In math intensive programs, this is visually annoying and unnatural. For this reason, the math and complex number libraries violate the naming convention in favor of familiar names. To take the sine of angle **`a#`**, therefore, write:

```
x# = SIN(a#)       ' correct
x# = XmaSin(a#)    ' wrong
```

Arguments and Return Values

Except for **`POWER()`**, all math library functions take one **`DOUBLE`** argument and return a **`DOUBLE`** result. **`POWER()`** takes two **`DOUBLE`** arguments and returns a **`DOUBLE`** result.

Angles

Angles are always radians, for both arguments and return values.

To convert degrees to radians, multiply by **`$$DEGTORAD`**.
To convert radians to degrees, multiply by **`$$RADTODEG`**.

Declarations

Your program must contain **`EXTERNAL FUNCTION`** statements in the **`PROLOG`** for every math library function it calls. You can include all the function declarations and constant definition of the math library with an **`IMPORT "xma"`** in your prolog as in:

```
IMPORT  "xma"     ' include math library declarations
IMPORT  "xst"     ' include standard library declarations
```

Math Library Functions - Summary

SIN()	sine
COS()	cosine
TAN()	tangent
COT()	cotangent
SEC()	secant
CSC()	cosecant
ASIN()	arc-sine
ACOS()	arc-cosine
ATAN()	arc-tangent
ACOT()	arc-cotangent
ASEC()	arc-secant
ACSC()	arc-cosecant
SINH()	hyperbolic sine
COSH()	hyperbolic cosine
TANH()	hyperbolic tangent
COTH()	hyperbolic cotangent
SECH()	hyperbolic secant
CSCH()	hyperbolic cosecant
ASINH()	hyperbolic arc-sine
ACOSH()	hyperbolic arc-cosine
ATANH()	hyperbolic arc-tangent
ACOTH()	hyperbolic arc-cotangent
ASECH()	hyperbolic arc-secant
ACSCH()	hyperbolic arc-cosecant
LOG()	base e logarithm (natural log)
LOG10()	base 10 logarithm
EXP()	base e "anti-log" (e to the x)
EXP10()	base 10 "anti-log" (10 to the x)
SQRT()	square root
POWER()	power (x to the y)

Complex Number Function Library

Complex Number Library Functions

```
DOUBLE   = DCABS       (DCOMPLEX z)
DCOMPLEX = DCACOS      (DCOMPLEX z)
DOUBLE   = DCARG       (DCOMPLEX z)
DCOMPLEX = DCASIN      (DCOMPLEX z)
DCOMPLEX = DCATAN      (DCOMPLEX z)
DCOMPLEX = DCCONJ      (DCOMPLEX z)
DCOMPLEX = DCCOS       (DCOMPLEX z)
DCOMPLEX = DCCOSH      (DCOMPLEX z)
DCOMPLEX = DCEXP       (DCOMPLEX z)
DCOMPLEX = DCLOG       (DCOMPLEX z)
DCOMPLEX = DCLOG10     (DCOMPLEX z)
DOUBLE   = DCNORM      (DCOMPLEX z)
DCOMPLEX = DCPOLAR     (DOUBLE magnitude, DOUBLE angle)
DCOMPLEX = DCPOWERCC   (DCOMPLEX z, DCOMPLEX n)
DCOMPLEX = DCPOWERCR   (DCOMPLEX z, DOUBLE n)
DCOMPLEX = DCPOWERRC   (DOUBLE z, DCOMPLEX n)
DCOMPLEX = DCRMUL      (DCOMPLEX x, DOUBLE y)
DCOMPLEX = DCSIN       (DCOMPLEX z)
DCOMPLEX = DCSINH      (DCOMPLEX z)
DCOMPLEX = DCSQRT      (DCOMPLEX z)
DCOMPLEX = DCTAN       (DCOMPLEX z)
DCOMPLEX = DCTANH      (DCOMPLEX z)

SINGLE   = SCABS       (SCOMPLEX z)
SCOMPLEX = SCACOS      (SCOMPLEX z)
SINGLE   = SCARG       (SCOMPLEX z)
SCOMPLEX = SCASIN      (SCOMPLEX z)
SCOMPLEX = SCATAN      (SCOMPLEX z)
SCOMPLEX = SCCONJ      (SCOMPLEX z)
SCOMPLEX = SCCOS       (SCOMPLEX z)
SCOMPLEX = SCCOSH      (SCOMPLEX z)
SCOMPLEX = SCEXP       (SCOMPLEX z)
SCOMPLEX = SCLOG       (SCOMPLEX z)
SCOMPLEX = SCLOG10     (SCOMPLEX z)
SINGLE   = SCNORM      (SCOMPLEX z)
SCOMPLEX = SCPOLAR     (SINGLE magnitude, SINGLE angle)
SCOMPLEX = SCPOWERCC   (SCOMPLEX z, SCOMPLEX n)
SCOMPLEX = SCPOWERCR   (SCOMPLEX z, SINGLE n)
SCOMPLEX = SCPOWERRC   (SINGLE z, SCOMPLEX n)
SCOMPLEX = SCRMUL      (SCOMPLEX x, SINGLE y)
SCOMPLEX = SCSIN       (SCOMPLEX z)
SCOMPLEX = SCSINH      (SCOMPLEX z)
SCOMPLEX = SCSQRT      (SCOMPLEX z)
SCOMPLEX = SCTAN       (SCOMPLEX z)
SCOMPLEX = SCTANH      (SCOMPLEX z)
```

Standard Function Library

Portability

The *standard function library* is a collection of popular functions. The standard function library is available on every implementation and its functions behave identically.

Recent Additions

New functions are added to the standard library on an irregular basis. The standard library you have may be ahead of this documentation.

Return Type and Arguments

To review the functions in your standard library, select **`HelpStandardLibrary`** in the main window pulldown menu to display **`xst.dec`** in the *InstantHelp* window. This is the most reliable and up to date information on standard library functions, because it's the current standard library prolog.

Pass by Reference - `@variable`

Many arguments in the function table have a `@` pass by reference prefix. These arguments fall into one or more of these catagories:

- *The language requires this argument be passed by reference (arrays).*
- *The value is modified intentionally by the function to return a value.*
- *The string value is not changed and pass by reference is faster.*

Numeric arguments with the pass by reference prefix return a value. Programs that don't need a particular numeric argument return value can pass that argument by value to increase speed.

Array and string arguments are not modified unless such modification is a purpose of the function. For example, in **`XstCopyArray (@`*array$[]*`, @`*copy$[]*`)`**, ***array$[]*** is not modified, but ***copy$[]*** is.

Unless otherwise stated, functions return non-zero to indicate error.

Composite Types and Constants

Several data types and constants defined in the standard library to support the standard library functions. For example, **`FILEINFO`** is a data type that supports **`XstGetFilesAndAttributes()`**.

A large number of constants are defined by the standard library, not only for standard library functions, but for intrinsic functions too. Almost all programs import the standard library with **`IMPORT "xst"`**. Some of the constants defined in the standard library are: file modes for **`OPEN()`**, drive types, file attributes, find modes, sort modes, data types returned by **`TYPE()`**, error and exception numbers, etc.

Standard Library Functions - Summary

```
*****  System Functions  *****

Xst                            ( )
XstVersion$                    ( )
XstCauseException              ( exception )
XstErrorNameToNumber           ( @error$, @error )
XstErrorNumberToName           ( error, @error$ )
XstExceptionNameToNumber       ( @exception$, @exception )
XstExceptionNumberToName       ( exception, @exception$ )
XstFileToSystemFile            ( fileNumber, @systemFileNumber )
XstGetApplicationEnvironment   ( @standalone, @reserved )
XstGetCommandLineArguments     ( @argc, @argv$[] )
XstGetConsoleGrid              ( @grid )
XstGetCPUName                  ( @cpu$ )
XstGetDateAndTime              ( @year, @month, @day, @weekDay, @hour, @minute, @second, @msec )
XstGetEndian                   ( @endian$$ )
XstGetEndianName               ( @endian$ )
XstGetEnvironmentVariable      ( @name$, @value$ )
XstGetEnvironmentVariables     ( @count, @envp$[] )
XstGetException                ( @exception )
XstGetExceptionFunction        ( @function )
XstGetOSName                   ( @name$ )
XstGetOSVersion                ( @major, @minor )
XstGetOSVersionName            ( @version$ )
XstGetPrintTab                 ( @pixels )
XstGetSystemError              ( @error )
XstGetSystemTime               ( @msec )
XstKillTimer                   ( timer )
XstSetCommandLineArguments     ( argc, @argv$[] )
XstSetDateAndTime              ( year, month, day, weekDay, hour, minute, second, msec )
XstSetEnvironmentVariable      ( @name$, @value$ )
XstSetExceptionFunction        ( function )
XstSetPrintTab                 ( pixels )
XstSetSystemError              ( sysError )
XstSleep                       ( msec )
XstStartTimer                  ( @timer, count, msec, callFunc )
XstSystemErrorToError          ( sysError, @error )
XstSystemErrorNumberToName     ( sysError, @sysError$ )
XstSystemExceptionNumberToName ( sysException, @sysException$ )
XstSystemExceptionToException  ( sysException, @exception )


*****  File Functions  *****

XstBinRead                     ( fileNumber, bufferAddress, maxBytes )
XstBinWrite                    ( fileNumber, bufferAddress, numBytes )
XstChangeDirectory             ( @directory$ )
XstCopyFile                    ( @sourceFile$, @destFile$ )
XstDeleteFile                  ( @filename$ )
XstGetCurrentDirectory         ( @directory$ )
XstGetDrives                   ( @count, @drive$[], @driveType[], @driveType$[] )
XstGetFileAttributes           ( @filename$, @attributes )
XstGetFiles                    ( @filter$, @files$[] )
XstGetFilesAndAttributes       ( @filter$, attributeFilter, @files$[], FILEINFO @info[] )
XstGetPathComponents           ( @file$, @path$, @drive$, @dir$, @filename$, @attributes )
XstGuessFileName               ( @old$, @new$, @guess$, @attributes )
XstLoadString                  ( @filename$, @text$ )
XstLoadStringArray             ( @filename$, @text$[] )
XstLockFileSection             ( fileNumber, mode, offset$$, length$$ )
XstMakeDirectory               ( @directory$ )
XstRenameFile                  ( @old$, @new$ )
XstSaveString                  ( @filename$, @text$ )
XstSaveStringArray             ( @filename$, @text$[] )
XstSaveStringArrayCRLF         ( @filename$, @text$[] )
XstSetCurrentDirectory         ( @directory$ )
XstUnlockFileSection           ( fileNumber, mode, offset$$, length$$ )
```

```
*****  String Functions  *****

XstBackArrayToBinArray             ( @backArray$[], @binArray$[] )
XstBackStringToBinString$          ( @rawString$ )
XstBinArrayToBackArray             ( @binArray$[], @backArray$[] )
XstBinStringToBackString$          ( @rawString$ )
XstBinStringToBackStringNL$        ( @rawString$ )
XstCopyArray                       ( @ANY[], @ANY[] )
XstDeleteLines                     ( @array$[], start, count )
XstFindArray                       ( mode, @text$[], @find$, line, pos, reps, skip, matches[] )
XstMultiStringToStringArray        ( @s$, @s$[] )
XstNextCField$                     ( sourceAddr, @index, @done )
XstNextCLine$                      ( sourceAddr, @index, @done )
XstNextField$                      ( @source$, @index, @done )
XstNextLine$                       ( @source$, @index, @done )
XstPathString$                     ( path$ )
XstReplaceArray                    ( mode, @text$[], @find$, @replace$, line, pos, reps, skip )
XstReplaceLines                    ( @dest$[], @source$[], firstD, countD, firstS, countS )
XstSetNewline                      ( @text$, newline )
XstStringArraySectionToString      ( @text$[], @copy$, x1, y1, x2, y2, term )
XstStringArraySectionToStringArray ( @text$[], @copy$[], x1, y1, x2, y2 )
XstStringArrayToString             ( @s$[], @s$ )
XstStringArrayToStringCRLF         ( @s$[], @s$ )
XstStringToNumber                  ( @s$, startOff, afterOff, rtype, value# )
XstStringToStringArray             ( @s$, @s$[] )


*****  Miscellaneous  *****

XstCompareStrings                  ( @addrString1, op, addrString2, flags )
XstQuickSort                       ( ANY x[], n[], low, high, flags )
```

`Xst()`	**`Xst ( )`** Initialize the standard function library. Every program must call this function before it calls any other standard library function. **`Xst()`** can be called any number of times without adverse effects.
`XstVersion$()`	*version$* **`= XstVersion$ ( )`** Return a string containing the standard function library version.
`XstCauseException()`	**`XstCauseException (`***exception***`)`** Cause the specified exception. ***exception*** is the native exception number, not the system exception number.
`XstErrorNameToNumber()`	**`XstErrorNameToNumber (`***error$***`, @`***error***`)`** Convert the one or two part error name in ***error$*** into an ***error*** number. See **`xst.dec`** for **`$$ErrorObject`** and **`$$ErrorNature`** constants.
`XstErrorNumberToName()`	**`XstErrorNumberToName (`***error***`, @`***error$***`)`** Convert the one or two part ***error*** number into an ***error$*** name.
`XstExceptionNameToNumber()`	**`XstExceptionNameToNumber (`***exception$***`, @`***exception***`)`** Convert a native ***exception$*** name into a native ***exception*** number.
`XstExceptionNumberToName()`	**`XstExceptionNumberToName (`***exception***`, @`***exception$***`)`** Convert a native ***exception*** number into a native ***exception$*** name.
`XstFileToSystemFile()`	**`XstFileToSystemFile (`***filenumber***`, @`***systemFilenumber***`)`** Convert a native ***filenumber*** returned by **`OPEN()`** into the ***systemFilenumber*** - the file number or handle the operating system refers to the file with. This makes it possible to call operating system functions directly to get information about the file.
`XstGetApplicationEnvironment()`	**`XstGetApplicationEnvironment (@`***standalone***`, @`***reserved***`)`** Return a ***standalone*** variable to tell whether the program is currently running as a standalone executable as opposed to in the environment.
`XstGetCommandLineArguments()`	**`XstGetCommandLineArguments (@`***argCount***`, @`***argv$***`[])`** Return the number of command line arguments in ***argCount***, and the command line argument strings in ***argv$[]***. ***argCount*** should never be `0` or less, since the name of the program is the first argument, unless **`XstSetCommandLineArguments()`** has changed them.

	Call **`XstGetCommandLineArguments()`** with (***argCount*** **`< 0`**) to get the original ***argCount*** and ***argv$[]*** in the event they have been changed by **`XstSetCommandLineArguments()`**.
`XstGetConsoleGrid()`	**`XstGetConsoleGrid (@`*grid*`)`** Return the grid number of the default console grid in ***grid***. *GuiPrograms* usually do not input or display information with the console window, and its presence on the display is superfluous. Therefore, many programs send a **`HideWindow`** or **`DestroyWindow`** message to the console grid to remove it from the display.
`XstGetCPUName()`	**`XstGetCPUName (@`*name$*`)`** Return the generic name of the central processor unit in ***name$***.
`XstGetDateAndTime()`	`XstGetDateAndTime (@`*year*`, @`*month*`, @`*day*`, @`*weekDay*`, @`*hour*`, @`*min*`, @`*sec*`, @`*msec*`)` Get the current date and time in GMT (Greenwich mean time). **`weekDay`** refers to the day of the week, as in Sunday, Monday, etc, and may not be available on some systems.
`XstGetEndian()`	**`XstGetEndian (@`*endian$$*`)`** Return a 64-bit endian descriptor that contains the following 8 bytes: **`0x00`**, **`0x01`**, **`0x02`**, **`0x03, 0x04, 0x05, 0x06, 0x07`** in the lowest to highest addresses of ***endian$$***. The value of ***endian$$*** is therefore **`0x0706050403020100`** on little endian systems, and **`0x0001020304050607`** on pure big endian systems.
`XstGetEndianName()`	**`XstGetEndianName (@`*endian$*`)`** Return **`"LittleEndian"`** or **`"BigEndian"`** in ***endian$***.
`XstGetEnvironmentVariable()`	**`XstGetEnvironmentVariable (@`*name$*`, @`*value$*`)`** Get the string ***value$*** of the environment variable with called ***name$***. For example, **`XstGetEnvironmentVariable (@"PATH", @path$)`**.
`XstGetEnvironmentVariables()`	**`XstGetEnvironmentVariables (@`*count*`, @`*envp$*`[])`** Return the number of environment variable strings in count, and the environment variable strings in ***envp$[]***. The strings contain both the name of the environment variable and its value, separated by an "=", as in **`"PATH=c:\windows; c:\windows\system; c:\xb"`**.
`XstGetException()`	**`XstGetException (@`*exception*`)`** Return the native ***exception*** number of the most recent exception.
`XstGetExceptionFunction()`	**`XstGetExceptionFunction (@`*functionAddress*`)`** Get the address of the current exception function in ***functionAddress***.

	When an exception occurs in a standalone program, the exception function established by **`XstSetExceptionFunction()`** is executed.
`XstGetOSName()`	**`XstGetOSName (@`*`name$`*`)`** Return the operating system name in ***name$***. Examples include "**`Windows`**", "**`WindowsNT`**", "**`UNIX`**", "`OS2`". See **`XstGetVersion()`**.
`XstGetOSVersion()`	**`XstGetOSVersion (@`*`major`*`, @`*`minor`*`)`** Return the ***major*** and ***minor*** portions of the operating system version. The ***major*** and ***minor*** part are the integer and fractional portions of the complete version number, so version 3.10 of the Windows operating system, **`major = 0x0003`** and **`minor = 0x000A`**.
`XstGetOSVersionName()`	**`XstGetOSVersionName (@`*`version$`*`)`** Return the operating system ***version$*** number.
`XstGetPrintTab()`	**`XstGetPrintTab (@`*`pixels`*`)`** Return the number of ***pixels*** between tab positions in the console.
`XstGetSystemError()`	**`XstGetSystemError (@`*`sysError`*`)`** Return the most recent operating system ***sysError*** number.
`XstGetSystemTime()`	**`XstGetSystemTime (@`*`msec`*`)`** Return the value of the free running time in ***msec***.
`XstKillTimer()`	**`XstKillTimer (`*`timer`*`)`** Kill the specified ***timer***.
`XstSetCommandLineArguments()`	**`XstSetCommandLineArguments (`*`argCount`*`, @`*`argv$`*`[])`** Set the number of command line arguments to ***argCount***, and the command line argument strings to ***argv$[]***. ***argCount*** should never be less than `0`.
`XstSetDateAndTime()`	**`XstSetDateAndTime (`*`year`*`, `*`month`*`, `*`day`*`, `*`weekDay`*`, `*`hour`*`, `*`min`*`, `*`sec`*`, `*`msec`*`)`** Set the current system date and time. This function may fail if the user running the task does not have supervisor or administrator priority.
`XstSetEnvironmentVariable()`	**`XstSetEnvironmentVariable (@`*`name$`*`, @`*`value$`*`)`** Set environment variable ***name$*** to ***value$***. For example, **`XstSetEnvironmentVariable (@"PATH", @"c:\windows")`**.

`XstSetExceptionFunction()`	**`XstSetExceptionFunction (`***functionAddress***`)`** Set the exception function to ***functionAddress***. When exceptions occur in standalone programs, the exception function is executed. The exception function must take zero arguments.
`XstSetPrintTab()`	**`XstSetPrintTab (`***pixels***`)`** Set the number of ***pixels*** between tab positions in the console.
`XstSetSystemError()`	**`XstSetSystemError (`***error***`)`** Set the current operating system ***error*** number.
`XstSleep()`	**`XstSleep (`***msec***`)`** Suspend program execution for ***msec*** milliseconds. While a program sleeps, other programs get an opportunity to run.
`XstStartTimer()`	**`XstStartTimer (@`***timer***`, `***count***`, `***msec***`, `***function***`)`** Create a ***timer***, set its cycle ***count***, set its ***msec*** countdown time, set its four argument timeout ***function*** **`()`** address, and start the timer. Each time the ***timer*** times out, **`XstStartTimer()`** calls: **`@`*****function*** **`(`*****timer***`, @`***count***`, `***msec***`, `***time***`)`** ***function*** **`()`** can kill the timer in the following ways: return **`-1`** set **`count = 0`** set **`count = -1`** call **`XstKillTimer (timer)`** ***function*** **`()`** must accept four **`XLONG`** arguments, and can change ***count*** to change the number of timeout cycles remaining.
`XstSystemErrorToError()`	**`XstSystemErrorToError (`***sysError***`, @`***error***`)`** Convert an operating system error number to a native ***error*** number.
`XstSystemErrorNumberToName()`	**`XstSystemErrorNumberToName (`***error***`, @`***error$***`)`** Convert a system ***error*** number into an ***error$*** name string.
`XstSystemExceptionNumberToName()`	**`XstSystemExceptionNumberToName (`***sysException***`, @`***sysException$***`)`** Convert an operating system exception number into a string name.
`XstSystemExceptionToException()`	**`XstSystemExceptionToException (`***sysException***`, @`***exception***`)`** Convert an operating system exception into a native ***exception***.

XstBinRead()

bytesRead **= XstBinRead (***fileNumber***,** *address***,** *maxBytes***)**

Read binary data from diskfile into memory.

```
bytesRead    = number of bytes read into memory
fileNumber   = file number returned by OPEN()
address      = memory address to read file data into
maxBytes     = maximum number of bytes to read
```

XstBinRead() reads up to ***maxBytes*** into memory at address from ***fileNumber***, starting at the current value of the file pointer.

If fewer than ***maxBytes*** exist between the current file pointer and the end of file, all remaining bytes are read in. The number of bytes read into memory is returned in ***bytesRead*** unless an error occurs, in which case ***bytesRead*** contains **-1** and **$$XERROR** contains the runtime error number.

An error is returned if a disk access error occurs, ***fileNumber*** is not open for reading, or the file pointer is at or beyond the end of file.

The **READ** statement is more efficient, safer, and usually more appropriate than **XstBinRead()**. **READ** never reads too much data, thereby writing outside the target variable. **XstBinRead()** will attempt to read any quantity of data into any address. Therefore, it can write data outside the appropriate area, which almost always leads to fatal memory faults that crash the program and the development environment.

READ only works with variables, strings, and arrays that are part of the language, however. When C library functions supply an address to receive data, **XstBinRead()** is an appropriate choice.

XstBinWrite()

error **= XstBinWrite (***fileNumber***,** *address***,** *writeBytes***)**

Write binary data to diskfile from memory.

```
error        = non-zero if an error occurred
fileNumber   = file number returned by OPEN()
address      = memory address to get data from
writeBytes   = number of bytes to write to file
```

XstBinWrite() writes ***writeBytes*** from memory at ***address*** to ***fileNumber***, starting at the current value of the file pointer.

0 is returned in ***error*** unless an error occurred, in which case **##ERROR** contains the runtime error number.

An error is returned if a disk access error occurs, or ***fileNumber*** is not open for writing.

The **WRITE** statement is more efficient and usually more appropriate than **XstBinWrite()**. **WRITE** only works with variables, strings,

	and arrays that are part of the language, however. When C library functions supply an address of data to be saved, **`XstBinWrite()`** is an appropriate choice.
`XstChangeDirectory()`	*error* **`= XstChangeDirectory (`***directory$***`)`** Change the default or working directory to ***directory$***.
`XstCopyFile()`	*error* **`= XstCopyFile (`***sourceFilename$***`,`** ***newFilename$***`)`** Create a new file called ***newFilename$*** and copy the contents of the existing ***sourceFilename$*** into ***newFilename$***.
`XstDeleteFile()`	*error* **`= XstDeleteFile (`***filename$***`)`** Delete the specified ***filename$***.
`XstGetCurrentDirectory()`	*error* **`= XstGetCurrentDirectory (@`***directory$***`)`** Get the current default aka working directory name in ***directory$***.
`XstGetDrives()`	*error* **`= XstGetDrives (@`***count***`, @`***drive$[]***`, @`***type[]***`, @`***type$[]***`)`** Get the drives currently recognized by the system, where ***count*** contains the number of drives, ***drive$[]*** contains their names, ***type[]*** contains a drive type, and ***type$[]*** contains the name of the drive type. The standard library defines drive type constants - see **`xst.dec`**. Note that UNIX systems present drives as directories, so drives are invisible.
`XstGetFileAttributes()`	*error* **`= XstGetFileAttributes (@`***filename$***`, @`***attributes***`)`** Get the file attributes of the specified ***filename$***. The standard library defines file attribute constants - see **`xst.dec`**.
`XstGetFiles()`	*maxLength* **`= XstGetFiles (@`***filter$***`, @`***file$[]***`)`** Get the array of file names in ***file$[]*** that corresponds to the filename ***filter$*** string. ***filter$*** can contain drive, path, and filename with "**`*`**" and "**`?`**" wildcard characters.
`XstGetFilesAndAttributes()`	*maxLen* **`= XstGetFilesAndAttributes (@`***filter$***`, @`***filter***`, @`***file$[]***`, FILEINFO @`***info[]***`)`** Get an array of filenames in ***file$[]*** and file information in ***info[]*** for the files specified by the drive/path/filename in ***filter$*** and the file attributes in ***filter***. The ***info[]*** array is type **`FILEINFO`**, as defined in "**`xst.dec`**". The number of characters in the longest filename is returned in ***maxLen***.
`XstGetPathComponents()`	**`(@`***file$***`, @`***path$***`, @`***drive$***`, @`***dir$***`, @`***filename$***`, @`***attributes***`)`** Get the components of a ***file$***. The ***path$***, ***drive$***, ***dir$***, and ***filename$***, and ***attributes*** of the specified file are returned.

`XstGuessFileName()`	**`XstGuessFileName (@`*`old$`*`, @`*`new$`*`, @`*`guess$`*`, @`*`attributes`*`)`**
`XstLoadString()`	*error* **`= XstLoadString (@`*`filename$`*`, @`*`string$`*`)`** Load the contents of ***filename$*** into a ***string$***. The length of ***string$*** is the same as the number of bytes in ***filename$***. ***string$*** can contain any combination of ascii and/or binary bytes.
`XstLoadStringArray()`	*error* **`= XstLoadStringArray (@`*`filename$`*`, @`*`string$[]`*`)`** Load the contents of ***filename$*** into string array ***string$[]***. The contents of filename$ are broken into separate "lines" by any of the following newline byte sequences - **`"\r\n"`**, **`"\n\r"`**, **`"\n"`**. The newline bytes are not put into ***string$[]***. If the last characters in ***filename$*** are a newline byte sequence, the last element of ***string$[]*** is an empty string aka `""`.
`XstLockFileSection()`	*error* **`= XstLockFileSection (`*`filenumber`*`, `*`mode`*`, `*`offset$$`*`, `*`length$$`*`)`**
`XstMakeDirectory()`	*error* **`= XstMakeDirectory (@`*`directory$`*`)`**
`XstRenameFile()`	*error* **`= XstRenameFile (@`*`oldName`*`$, @`*`newName`*`$)`**
`XstSaveString()`	*error* **`= XstSaveString (@`*`filename$`*`, @`*`text$`*`)`**
`XstSaveStringArray()`	*error* **`= XstSaveStringArray (@`*`filename$`*`, @`*`text$[]`*`)`**
`XstSaveStringArrayCRLF()`	*error* **`= XstSaveStringArrayCRLF (@`*`filename$`*`, @`*`text$[]`*`)`**
`XstSetCurrentDirectory()`	*error* **`= XstSetCurrentDirectory (@`*`directory$`*`)`**
`XstUnlockFileSection()`	*error* **`= XstUnlockFileSection (`*`filenumber`*`, `*`mode`*`, `*`offset$$`*`, `*`length$$`*`)`**

`XstBackArrayToBinArray()`	*error* `= XstBackArrayToBinArray (@`***back$[]***`, @`***bin$[]***`)` Make a duplicate of ***back$[]*** in ***bin$[]*** with all backslash characters converted into their binary equivalents. For example, every occurance of two character sequence "`\t`" in ***back$[]*** into a single `0x09` "tab" character in ***bin$[]***.
`XstBackStringToBinString$()`	*error* `= XstBackStringToBinString (@`***back$***`, @`***bin$***`)` Make a duplicate of ***back$*** in ***bin$*** with all backslash characters converted to their binary equivalents. For example, convert every occurrence of two character sequence "`\t`" in ***back$*** into a single `0x09` "tab" character in ***bin$***.
`XstBinArrayToBackArray()`	*error* `= XstBinArrayToBackArray (@`***bin$[]***`, @`***back$[]***`)` Make a duplicate of ***bin$[]*** in ***back$[]*** with all `0x00-0x1F` and `0x80-0xFF` characters converted to backslash character equivalents. For example, convert every one byte `0x09` "tab" character in ***bin$[]*** to the two character backslash character sequence "`\t`" in ***back$[]***.
`XstBinStringToBackString$()`	*error* `= XstBinStringToBackString (@`***bin$***`, @`***back$***`)` Make a duplicate of ***bin$*** in ***back$*** with every `0x00-0x1F` and `0x80-0xFF` character converted to backslash character equivalent. For example, convert every one byte `0x09` "tab" character in ***bin$*** to the two character backslash character sequence "`\t`" in ***back$***.
`XstBinStringToBackStringNL$()`	*error* `= XstBinStringToBackStringNL (@`***bin$***`, @`***back$***`)` Same as `XstBinStringToBackString()` except `0x0A` newline characters are not converted into their backslash character equivalent.
`XstCopyArray()`	*error* `= XstCopyArray (@`***array[]***`, @`***copy[]***`)` Return a copy of simple numeric type or string ***array[]*** in ***copy[]***. `XstCopyArray()` cannot copy composite arrays, which includes `SCOMPLEX` and `DCOMPLEX` arrays, as well as all user-defined and composite type arrays. Make sure ***copy[]*** is the same type as ***array[]***.
`XstDeleteLines()`	*error* `= XstDeleteLines (@`***text$[]***`, `***first***`, `***count***`)` Delete ***count*** lines from string array ***text$[]*** starting at line ***first***.
`XstFindArray()`	`XstFindArray (`***mode***`, @`***text$[]***`, @`***find$***`, @`***line***`, @`***pos***`, @`***match***`)` `XstFindArray()` looks for a ***find$*** string within text array ***text$[]*** starting at ***line***, ***pos***. `text$[0]` is ***line*** 0 and the first character on each line is ***pos*** 0. If an `XstFindArray()` finds an occurance of ***find$*** in ***text$[]*** given

the instructions in the ***mode*** argument, ***match*** is assigned a non-zero value and ***line***, ***pos*** are assigned the line and character position of the first character of the string in ***text$[]*** that matched ***find$***.

`XstFindArray()` does not alter ***text$[]*** or ***find$***.

mode`=0` tells `XstFindArray()` to do a forward, case-sensitive find. To control the find, `OR` together mode constants from `xst.dec` :

```
$$FindForward
$$FindReverse
$$FindDirection
$$FindCaseSensitive
$$FindCaseInsensitive
$$FindCaseSensitivity
```

`XstMultiStringToStringArray()`

`XstMultiStringToStringArray (@`*string$*`, @`*array$[]*`)`

`XstMultiStringToStringArray()` converts a ***string$*** into a string ***array$[]*** by breaking the string into separate strings at each occurance of an `\r` character. Note that the line separator character is not the `\n` aka newline character, and that the lines in ***array$[]*** may therefore contain `\n` characters. `\r` characters are discarded.

`XstNextCField$()`

string$ **`= XstNextCField$ (`*address*`, @`*index*`, @`*done*`)`**

Return the next text element from a C string.

```
string$ = next text element from C string
address = memory address of C string
index   = character position in C string  ( 1st byte = 1 )
done    = end of C string reached
```

`XstNextCField$()` returns the next text element in the string at ***address***, starting at character position ***index***. ***index*** is advanced to the separator that terminates the text element. Text elements are separated by *bounding characters*, which are characters with a value `<= 0x20` (space, tab, newline, return, and all control characters) and characters with a value `>= 0x7F` (all special characters).

All bounding characters are skipped. Then valid text characters are collected in ***string$*** until a bounding character is found or the end of the string is reached, which is the first null character in the string.

index and ***done*** are normally passed by reference because useful information is returned in these variables. ***index*** is returned with the position of the character after the text element, and ***done*** is returned with a non-zero value if ***index*** entered with a value greater than the length of the string at ***address***.

If index and done are passed by reference, `XstNextCField$()` can be called repeatedly to read successive text elements from the string at ***address***.

If ***index*** `<= 0` is passed to `XstNextCField()`, it is set to `1`.

`XstNextCLine$()`

string$ **`= XstNextCLine$ (`*****address*****`, @`*****index*****`, @`*****done*****`)`**

Return the next newline terminated string from a C string.

```
address = memory address of C string
index   = character position in C string  ( 1st byte = 1 )
done    = end of C string reached
```

`XstNextCLine$()` returns the ***string*** in from the C string at ***address*** that starts at ***index*** and ends with the next newline character or null character (end of string), whichever comes first.

string$ is returned without the terminating newline or null character. ***index*** and ***done*** are normally passed by reference because useful information is returned in these variables. ***index*** is moved past the newline or end of string. ***done*** is returned with a non-zero value if the character at ***index*** is a null character.

If ***index*** and ***done*** are passed by reference, **`XstNextCLine$()`** can be called repeatedly to read successive lines from the C string at ***address***.

If ***index*** **`<= 0`** is passed to **`XstNextCLine()`**, it is set to **`1`**.

`XstNextField$()`

string$*** **`= XstNextField$ (@`*****source$**`, @`*****index*****`, @`*****done*****`)`**

Return the next text element from a string.

```
string$  = next text element from string
source$  = string to extract text element from
index    = character position in source$
done     = end of string reached
```

`XstNextField$()` returns the next text element from ***source$***, starting at character position ***index***. ***index*** is advanced to the separator that terminates the text element. Text elements are separated by *bounding characters*, which are characters with a value **`<= 0x20`** (space, tab, newline, return, and all control characters) and characters with a value **`>= 0x7F`** (all special characters).

All bounding characters are skipped. Then valid text characters are collected in ***string$*** until a bounding character is found or the end of the string is reached.

index and ***done*** are normally passed by reference because useful information is returned in these variables. ***index*** is returned with the position of the character after the text element, and ***done*** is returned with a non-zero value if ***index*** entered with a value greater than the length of ***source$***.

If ***index*** and ***done*** are passed by reference, **`XstNextField$()`** can be called repeatedly to read successive text elements from ***source$***.

If ***index*** **`<= 0`** is passed to **`XstNextField()`**, it is set to **`1`**.

source$ is not modified by **`XstNextField$()`**, so it can be passed by reference for optimal speed.

`XstNextLine$()`

string$*** **`= XstNextLine$ (@`*****source$**`, @`*****index*****`, @`*****done*****`)`**

Return the next newline terminated string from a string.

```
source$ = string to extract the next line from
index   = character position in C string  ( 1st byte = 1 )
done    = end of C string reached
```

`XstNextLine$()` returns the next ***string$*** from ***source$*** that starts at ***index*** and ends with the next newline character or end of string, whichever comes first.

string$ is returned without a terminating character. ***index*** and ***done*** are normally passed by reference because useful information is returned in these variables. ***index*** is moved past the newline or end of string. ***done*** is returned with a non-zero value if ***index*** is greater than the length of ***source$***.

If ***index*** and ***done*** are passed by reference, **`XstNextLine$()`** can be called repeatedly to read successive lines from ***source$***.

If ***index*** **`<= 0`** is passed to **`XstNextLine$()`**, it is set to **`1`**.

source$ is not modified by **`XstNextLine$()`**, so it can be passed by reference for optimal speed.

`XstPathString$`

path$*** **`= XstPathString$ (`*****path$**`)`**

`XstReplaceArray()`

`XstReplaceArray (`***mode*****`, @`*****text$[]*****`, @`*****find$*****`, @`*****replace$*****`, @`*****line*****`, @`*****pos*****`, @`*****match*****`)`**

`XstReplaceArray()` looks for a ***find$*** string within text array ***text$ []*** starting at ***line***, ***pos***. **`text$[0]`** is ***line*** 0 and the first character on each line is ***pos*** 0.

If an **`XstReplaceArray()`** finds an occurance of ***find$*** in ***text$[]*** given the instructions in the ***mode*** argument, ***match*** is assigned a non-zero value and ***line***, ***pos*** are assigned the line and character position of the first character of the string in ***text$[]*** that matched ***find$***, and the matched string in ***text$[]*** is replaced by ***replace$***.

`XstReplaceArray()` does not alter ***text$[]***, ***find$***, or ***replace$***.

mode=**`0`** tells **`XstReplaceArray()`** to find forward, case-sensitive. To control the find, **`OR`** together mode constants from **`xst.dec`** :

```
$$FindForward
$$FindReverse
$$FindDirection
$$FindCaseSensitive
$$FindCaseInsensitive
```

	$$FindCaseSensitivity
XstReplaceLines()	**XstReplaceLines** (@*d$[]*, @*s$[]*, *firstD*, *countD*, *firstS*, *countS*)
XstSetNewline()	**XstSetNewline** (@*text$*, *newline*)
XstStringArraySectionToString()	**XstStringArraySectionToString** (@*text$[]*, @*copy$*, *x1*, *y1*, *x2*, *y2*, *term*)
XstStringArraySectionToStringArray()	**XstStringArraySectionToStringArray** (@*text$[]*, @*copy$[]*, *x1*, *y1*, *x2*, *y2*)
XstStringArrayToString()	**XstStringArrayToString** (@*text$[]*, @*text$*)
XstStringArrayToStringCRLF()	**XstStringArrayToStringCRLF** (@*text$[]*, @*text$*)

XstStringToNumber()

specType = **XstStringToNumber** (@*value$*, *start*, @*after*, @*rtype*, @*value$$*)

Convert all or part of a string into a number of natural data type.

```
  specType   = explicit type  ( or -1 for numeric format error )
  start      = starting offset in value$  ( not modified )
  after      = returned with offset after last numeric character
  rtype      = returned with "natural data type" of value
  value$$    = returned with value of number in rtype format
```

XstStringToNumber() converts all or part of **value$** into a numeric value. It returns the numeric value in **value$$**, its natural type in **rtype**, and any explicit type in **specType**. **value$** can be passed by reference for faster execution.

XstStringToNumber() scans **value$** from offset **startOff**, skips leading whitespace and unprintable characters, then converts subsequent characters into a number.

If the first of the subsequent characters cannot begin a valid number, **XstStringToNumber()** returns **specType=-1**, **rtype=0**, and the offset of the bad character in **afterOff**.

XstStringToNumber() collects characters until it encounters one that is not a valid part of a number. It returns the offset of this character in **afterOff**, the natural type of the number in **rtype**, and the value of the number in **value$$**.

value$$ is a **GIANT** number, but the numeric value stored in **value$$** is not in **GIANT** format unless **rtype=$$GIANT**.

rtype is always **SLONG, XLONG, GIANT, SINGLE,** or **DOUBLE**. The final return value can be extracted from **value$$** as follows:

```
SELECT CASE rtype
  CASE $$SLONG  : value  = GLOW(value$$)
  CASE $$XLONG  : value  = value$$
  CASE $$SINGLE : value! = SMAKE(GLOW(value$$))
  CASE $$DOUBLE : value# = DMAKE(GHIGH(value$$), GLOW(value$$))
END SELECT
```

If **specType=-1**, **rtype!=0**, an **rtype** was returned in **value$$**, but the format is suspect. Examples include:

```
0s7F8033jk          ' 8 hex digits required after "0s"
0d3FED0000000 hi    ' 16 hex digits required after "0d"
12.34d+8243         ' larger than largest number representable
```

If **specType = SLONG**, **XLONG**, **GIANT**, **SINGLE**, or **DOUBLE**, then **rtype=specType**, and the type was specified in the number. Examples of specified numeric types include:

```
0b1010010010111        ' XLONG:  "0b" that fits in 32-bits
0b1010...010111        ' XLONG:  "0b" that won't fit in 32-bits
0o361032723            ' XLONG:  "0o" that fits in 32-bits
0o7373315631277        ' XLONG:  "0o" that won't fit in 32-bits
0x12345678             ' XLONG:  "0x" followed by 0-8 hex digits
0x123456789AB          ' GIANT:  "0x" followed by 9+ hex digits
0s3F800000             ' SINGLE: "0s" followed by 8 hex digits
0d3FE0000000000000     ' DOUBLE: "0d" followed by 16 hex digits
```

XstStringToStringArray()

XstStringToStringArray (@*text$*, @*text$[]*)

`XstCompareStrings`

`XstCompareStrings (`*addrString1*`,` *op*`,` *addrString2*`,` *flags*`)`

`XstQuickSort()`

`XstQuickSort (@`*sortArray*`[], @`*orderArray*`[],` *first*`,` *last*`,` *flags*`)`

Sort the contents of all or part of an array.

```
sortArray[]   = array to sort all or part of
orderArray[]  = optional array left with original indices
first         = first element of region in sortArray[] to sort
last          = last element of region in sortArray[] to sort
flags         = see xst.dec file : OR appropriate flags together
                $$SortIncreasing     vs  $$SortDecreasing
                $$SortAlphabetic     vs  $$SortAlphaNumeric
                $$SortCaseSensitive  vs  $$SortCaseInsensitive
```

`XstQuickSort()` sorts the elements of **`sortArray[]`** between ***first*** and ***last***. Depending on ***flags***, the sorted elements are stored in increasing or decreasing order, are sorted alphabetic or alphanumeric, and sorted case sensitive or case insensitive.

The data type of **`sortArray[]`** can be any of the following:

```
SBYTE     UBYTE
SSHORT    USHORT
SLONG     ULONG      XLONG
GIANT
SINGLE    DOUBLE
STRING
```

If **`orderArray[]`** enters **`XstQuickSort()`** with no elements, it is ignored. Otherwise it is dimensioned to the same size as **`sortArray[]`**, filled with `0,1,2,3,4,5...`, then sorted in parallel with **`sortArray[]`**. When the sort is finished, it contains the original element number for every element in **`sortArray[]`**.

`sortArray[]` must be a one dimension array. **`orderArray[]`** must be a one dimensional **`XLONG`** array.

An error is generated if:
- ***last*** is less than ***first***.
- ***first*** or ***last*** is less than zero
- ***first*** or ***last*** is greater than the upper bound of **`sortArray[]`**

Depending on the type of **`sortArray[]`**, **`XstQuickSort()`** calls internal functions that sort **`SLONG`**, **`XLONG`**, **`GIANT`**, **`DOUBLE`**, or **`STRING`** arrays. When **`sortArray[]`** is another type, a temporary array of the next larger type from this selection is created, the contents of **`sortArray[]`** are transferred to the temporary array, the sort is performed, then the contents are transferred back into the original array.