API documentation for libcore

Contents

1 Overview

This is a standard system library which is used by all applications. It contains all basic routines necessary to write console programs.

2 Builtin types and functions.

2.1 Description

These types and functions are builtin and do not require any `use` statement.

2.2 Types

```
type Any;
```

Parent type of all other types.

```
type Byte < Any;
```

Byte value. Values of this type are signed integer numbers in range from −128 to 127.

```
type Short < Any;
```

Short integer value. Values of this type are signed integer numbers in range from -32768 to 32767.

```
type Char < Any;
```

Character value. Values of this type are unsigned integer numbers in range from 0 to 65535.

```
type Int < Any;
```

Integer value. Values of this type are signed integer numbers in range from -2^{31} to $2^{31}-1$.

```
type Long < Any;
```

Long integer value. Values of this type are signed integer numbers in range from -2^{63} to $2^{63}-1$.

```
type Float < Any;
```

Single precision floating point number.

```
type Double < Any;
```

Double precision floating point number.

```
type Bool < Any;
```

Boolean value. Values of this type are `true` and `false`.

```
type String < Any;
```

A sequence of characters.

```
type Array < Any;
```

Parent type or all array types.

```
type Structure < Any;
```

Parent type of all structure types.

```
type Function < Any;
```

Parent type of all functional types.

```
type Error < Any;
```

Error thrown in exceptional situations. See error.eh for details.

2.3 Functions

```
def Any.tostr(): String;
```

Returns string representation of the value. If value is:

- `String` - value itself is returned;
- `Int`, `Long`, `Float` or `Double` - string representation as a decimal number is returned;
- `Bool` - string ”true” or ”false” is returned;
- `StrBuf` - new string allocated with characters currently contained in the buffer;
- `Function` - the name of the function is returned;
- `Array` - the string containing all array elements, separated by commas, is returned;
- `null` - function returns string `"null"`.

It is safe to call this function for values of other types but returned values are platform dependant and you should not rely on them.

```
def String.len(): Int;
```

Returns length of the string.

```
def String.ch(at: Int): Char;
```

Returns character at given position in the string. You may also use square brackets, as with array.

```
def String.substr(from: Int, to: Int): String;
```

Returns substring of the string. Returned substring contains all characters in range `from..to-1`.

```
def chstr(ch: Char): String;
```

Returns string containing single character.

```
def acopy(src: Array, sofs: Int, dest: Array, dofs: Int, len: Int);
```

Copies *len* elements from array *src* starting at index *sofs* to array *dest* starting at index *dofs*.

```
def Structure.clone(): Structure;
```

Clones structure. The returned value is new structure of the same type with all fields copied from the old structure.

3 io.eh — Work with file system and input/output routines.

```
use "io.eh"
```

3.1 Description

This header defines routines to work with file system and also input/output through the byte streams.

3.1.1 Work with files

File paths in Alchemy OS are specified Unix-style, i.e. directories in the path are separated with slash (`'/'`). Paths may include elements *this directory* `"."` and *parent directory* `".."`. If path starts with slash it is an *absolute path* from the root of the file system hierarchy. Otherwise, path is *relative* to the current working directory. For example, `"file.txt"` is a file in current directory and `"../icon.png"` is a file in directory parent to current. Absolute and relative paths may be converted into each other by functions abspath and relpath. Current working directory may be obtained with get_cwd and changed with set_cwd.

Standard file system operations include:

- fcreate - create new file;
- mkdir - make new directory;
- fremove - remove file;
- exists - check whether specified file exists;
- is_dir - check whether specified file is directory;
- flist - get the directory listing;
- fcopy - copy file;
- fmove - move/rename file;
- fsize - get size of the file;
- fmodified - get date of the last modification of the file.

Alchemy OS distinguishes three file attributes: the permission to read the file, to write to the file, and to execute the file. Note, that not all attributes are supported on all file systems.

3.1.2 Input and output streams

Reading and writing of files is done through the input and output streams.

IStream is an object from which you can read bytes sequentially, it may be file, internet address, etc... To read data from file you need to open input stream:

```
var input = fopen_r("somefile.txt") // open file for reading
```

Then you can read data from *input* variable. This is done by read method, which returns byte value as a number from 0 to 255. If you have reached the end of the file, method returns -1 (you can use convenient constant EOF). The next program reads entire file and prints integer value of each byte it reads.

```
var b = input.read()
while (b != EOF) {
  println("Byte: "+b)
  b = input.read()
}
```

Instead of reading single bytes you can read a block of bytes with readarray, and there are more `read*` functions for binary files (dataio.eh) or text files (textio.eh).

To close file after reading and free associated resources use close.

```
input.close()
```

Similarly, OStream is an object in which you can write bytes. There are two ways to open file for writing - with fopen_w you rewrite file clearing its previous contents and with fopen_a you append new data to the end of the file.

```
var out = fopen_w("test.txt")        // open file for writing
var buf = new [Byte] {'H','i','!'}   // create byte array
out.writearray(buf, 0, buf.len)      // write bytes from array in the file
out.write('!')                       // write single byte in the file
out.flush()                          // save changes to the file
out.close()                          // and finally close it
```

3.1.3 "Standard" streams

Every running program has three streams that are already opened and ready for I/O operation:

- stdin is a standard program input, usually input from the terminal. You can use this stream to ask user to type something interactively.
- stdout is a standard program output, usually output on the terminal. You can use this stream to print information on the phone screen.
- stderr is an output for program error messages.

All input/output functions have shorter versions to work with these streams. So, to print something on the terminal instead of

```
stdout().println("something")
```

you may write just

```
println("something")
```

3.2 Constants

```
const EOF = -1;
```

Returned by input stream read methods when all data has already been read.

3.3 Types

```
type IStream < Any;
```

An input stream is a binary input source from which data can be read in bytes.

```
type OStream < Any;
```

An output stream is a binary output source to which data can be written in bytes.

```
type BArrayOStream < OStream;
```

Output stream that writes to byte buffer.

3.4 Functions

```
def stdin(): IStream;
```

Returns the default input stream associated with this process. Functions read, readarray and skip read from this stream. You may assign different stream for input using setin.

```
def stdout(): OStream;
```

Returns the default output stream associated with this process. Functions write, writearray, print, println and flush write to this stream. You may assign different stream for output using setout.

```
def stderr(): OStream;
```

Returns the default error stream associated with this process. You may assign different stream for error messages using seterr.

```
def setin(in: IStream);
```

Redefines default input stream for this process.

```
def setout(out: OStream);
```

Redefines default output stream for this process.

```
def seterr(err: OStream);
```

Redefines default error stream for this process.

```
def IStream.read(): Int;
```

Reads byte from the input stream as unsigned value in range 0..255. If stream has reached its end this function returns EOF.

```
def IStream.readarray(buf: [Byte], ofs: Int, len: Int): Int;
```

Reads *len* bytes from the input stream and writes them to the byte array *buf* starting at index *ofs*. The function can end up in reading less bytes if the end of the stream is reached or data is otherwise unavailable. Returns total number of actually read bytes. If no bytes can be read because stream has reached its end EOF is returned.

```
def IStream.readfully(): [Byte];
```

Reads until the end of the stream and return data in array.

`def IStream.skip(num: Long): Long;`

Skips over and discards `num` bytes of data from the input stream. The function may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly 0. Returns total number of actually skipped bytes.

`def IStream.available(): Int;`

Returns count of buffered bytes which can be read without blocking.

`def IStream.reset();`

Resets this stream to the initial state. If stream supports reset this method repositions it to the beginning of the input or other appropriate beginning state. If reset is not supported, this method raises ERR_IO error.

`def IStream.close();`

Closes this stream and frees associated resources.

`def read(): Int;`

Reads byte from standard input as unsigned value in range 0..255. If stream has reached its end this function returns EOF.

`def readarray(buf: [Byte], ofs: Int, len: Int): Int;`

Reads *len* bytes from the standard input and writes them to the byte array *buf* starting at index *ofs*. The function can end up in reading less bytes if the end of the stream is reached or data is otherwise unavailable. Returns total number of actually read bytes. If no bytes can be read because stream has reached its end EOF is returned.

`def skip(num: Long): Long;`

Skips over and discards `num` bytes of data from the standard input. The function may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly 0. Returns total number of actually skipped bytes.

`def readline(): String;`

Reads a single line of text from standard input. UTF-8 encoding is used to decode string.

`def OStream.write(b: Int);`

Writes specified byte to the output stream.

```
def OStream.writearray(buf: [Byte], ofs: Int, len: Int);
```

Writes `len` bytes from given byte array starting from index `ofs` to the output stream.

```
def OStream.writeall(input: IStream);
```

Writes all data from given input to the output.

```
def OStream.print(a: Any);
```

Writes text representation of the given value to the output stream. UTF-8 encoding is used.

```
def OStream.println(a: Any);
```

Writes text representation of the given value to the output stream and terminates a line. UTF-8 encoding is used.

```
def OStream.flush();
```

Forces any buffered data to be written immediately.

```
def OStream.printf(fmt: String, args: [Any]);
```

Writes formatted string to the output stream using given format string and arguments. See String.format description on how formatting is used.

```
def OStream.close();
```

Closes this stream and frees associated resources.

```
def write(b: Int);
```

Writes specified byte to the standard output.

```
def writearray(buf: [Byte], ofs: Int, len: Int);
```

Writes `len` bytes from given byte array starting from index `ofs` to the standard output.

```
def print(a: Any);
```

Writes text representation of the given value to the standard output. UTF-8 encoding is used.

```
def println(a: Any);
```

Writes text representation of the given value to the standard output and terminates a line. UTF-8 encoding is used.

```
def flush();
```

Forces any buffered data in stdout to be written immediately.

```
def printf(fmt: String, args: [Any]);
```

Writes formatted string to the standard output using given format string and arguments. See String.format description on how formatting is used.

```
def pathfile(path: String): String;
```

Returns name of the file represented by this path.

```
def pathdir(path: String): String;
```

Returns parent directory for given path. For root directory this function returns `null`.

```
def abspath(path: String): String;
```

Converts given path to the absolute path. Pathnames `"."` and `"path/.."` are removed from result. If argument *path* is relative it is calculated from the current working directory.

```
def relpath(path: String): String;
```

Converts given path to the path relative to the current working directory. Pathnames `"."` and `"path/.."` are removed from result. If argument *path* is relative it is calculated from the current working directory.

```
def fcreate(path: String);
```

Creates new empty file in the filesystem.

```
def fremove(path: String);
```

Removes file or directory from the filesystem. Does nothing if file does not exist. This function can refuse to remove directory if it is not empty.

```
def mkdir(path: String);
```

Creates new empty directory in the filesystem.

```
def fcopy(source: String, dest: String);
```

Copies contents of file *source* to file *dest*. If *dest* does not exist then new file is created.

```
def fmove(source: String, dest: String);
```

Moves file to a new location and/or renames it.

```
def exists(path: String): Bool;
```

Tests whether the specified file exists.

```
def is_dir(path: String): Bool;
```

Returns `true` the specified file exists and is a directory.

```
def fopen_r(path: String): IStream;
```

Opens given file for reading. File must exist and be accessible for reading.

```
def fopen_w(path: String): OStream;
```

Opens given file for writing. If file exists then it must be accessible for writing. The previous contents of the file is erased. If file does not exist then new file is created in the filesystem.

```
def fopen_a(path: String): OStream;
```

Opens given file for appending. If file exists then it must be accessible for writing. New data is written to the end of the file. If file does not exist then new file is created in the filesystem.

```
def flist(dir: String): [String];
```

Returns file names that the specified directory contains. Directory names are ended with `'/'`. The pathnames `".."` and `"."` are not included in the return value. If the directory is empty then a zero-length array is returned.

```
def flistfilter(dir: String, pattern: String): [String];
```

Returns file names in the specified directory that match given pattern. Symbol `'?'` represents any character, symbol `'*'` represents any number of characters. For example, to get list of all files with `.txt` extension from the current directory:

```
var files = flistfilter("./", "*.txt")
```

If you need to search for characters `'?'` and `'*'`, escape them as `"\?"` and `"\*"`.

```
def fmodified(path: String): Long;
```

Returns time of the last modification of the file. To work with the returned value use functions from time.eh.

```
def fsize(path: String): Long;
```

Returns the size of the file in bytes. If size cannot be computed this function returns `-1L`.

```
def set_read(path: String, on: Bool);
```

Changes permission to read from the file. Does nothing if read flag is not supported by the filesystem.

```
def set_write(path: String, on: Bool);
```

Changes permission to write to the file. Does nothing if write flag is not supported by the filesystem.

```
def set_exec(path: String, on: Bool);
```

Changes permission to execute the file. Does nothing if exec flag is not supported by the filesystem.

```
def can_read(path: String): Bool;
```

Tests whether this file exists and has read permission.

```
def can_write(path: String): Bool;
```

Tests whether this file exists and has write permission.

```
def can_exec(path: String): Bool;
```

Tests whether this file exists and can be executed. Note, that native file system does not support exec flag, so this function always returns true.

```
def get_cwd(): String;
```

Gets the current working directory of the process.

```
def set_cwd(dir: String);
```

Sets the current working directory of the process. Directory must exist.

```
def space_total(root: String): Long;
```

Returns amount of total space in the file system at given mount point. Returns `-1L` if value is unknown.

```
def space_free(root: String): Long;
```

Returns amount of free space in the file system at given mount point. Returns `-1L` if value is unknown.

```
def space_used(root: String): Long;
```

Returns amount of used space in the file system at given mount point. Returns `-1L` if value is unknown.

```
def readurl(url: String): IStream;
```

Opens the specified URL for reading. URL must have the form `protocol://address`. Supported protocols as of release 2.0:

- `file:` — reads specified file;
- `http:` and `https:` — opens internet connection to read from the specified address;
- `res:` — reads system resource (intended to be used by system programs).

```
def matches_glob(file: String, pattern: String): Bool;
```

Tests if specified file name matches given pattern. Symbol `'?'` represents any character, symbol `'*'` represents any number of characters. If you need to match characters `'?'` and `'*'`, escape them as `"\?"` and `"\*"`.

```
def istream_from_ba(buf: [Byte]): IStream;
```

Creates new input stream that uses given byte array as source of data.

```
def BArrayOStream.new(): BArrayOStream;
```

Creates new output stream that writes to byte buffer.

```
def BArrayOStream.len(): Int;
```

Returns number of bytes written to this output stream so far.

```
def BArrayOStream.tobarray(): [Byte];
```

Allocates new byte array and copies current contents of this output stream in it.

```
def BArrayOStream.reset();
```

Resets this output stream discarding previous contents and setting length to zero.

4 dataio.eh — Additional I/O routines for binary data streams.

```
use "dataio.eh"
```

4.1 Description

This header enhances io.eh with I/O routines that allow reading and writing of various data types in binary form. Writing functions convert values in byte sequences that can be later read and converted back by corresponding reading functions.

If reading function meets EOF while decoding value, it raises an ERR_IO error.

4.2 Functions

```
def IStream.readbool(): Bool;
```

Reads one byte from the input and returns `true` if that byte is non-zero, `false` otherwise.

```
def IStream.readbyte(): Int;
```

Reads and returns one byte from the input as signed integer in range $-128..127$.

```
def IStream.readubyte(): Int;
```

Reads and returns one byte from the input as unsigned integer in range $0..255$.

```
def IStream.readshort(): Int;
```

Reads two bytes from the input and returns them as signed integer in range $-32768..32767$.

```
def IStream.readushort(): Int;
```

Reads two bytes from the input and returns them as unsigned integer in range $0..65535$.

```
def IStream.readint(): Int;
```

Reads four bytes from the input and returns them as `Int` value.

```
def IStream.readlong(): Long;
```

Reads eight bytes from the input and returns them as `Long` value.

```
def IStream.readfloat(): Float;
```

Reads four bytes from the input and decodes a `Float` value. It does so by reading an `Int` and decoding it with ibits2f.

```
def IStream.readdouble(): Double;
```

Reads eight bytes from the input and decodes a `Double` value. It does so by reading a `Long` and decoding it with lbits2d.

```
def IStream.readutf(): String;
```

Reads a string that has been encoded using a modified UTF-8 format. First, two bytes are read as unsigned integer. This value specifies are number of bytes that represent a string. The following bytes then decoded using ba2utf.

```
def readbool(): Bool;
```

Reads one byte from stdin and returns `true` if that byte is non-zero, `false` otherwise.

```
def readbyte(): Int;
```

Reads and returns one byte from stdin as signed integer in range $-128..127$.

```
def readubyte(): Int;
```

Reads and returns one byte from stdin as unsigned integer in range 0..255.

```
def readshort(): Int;
```

Reads two bytes from stdin and returns them as signed integer in range $-32768..32767$.

```
def readushort(): Int;
```

Reads two bytes from stdin and returns them as unsigned integer in range 0..65535.

```
def readint(): Int;
```

Reads four bytes from stdin and returns them as `Int` value.

```
def readlong(): Long;
```

Reads eight bytes from stdin and returns them as `Long` value.

`def readfloat(): Float;`

Reads four bytes from stdin and decodes a `Float` value. It does so by reading an `Int` and decoding it with ibits2f.

`def readdouble(): Double;`

Reads eight bytes from stdin and decodes a `Double` value. It does so by reading a `Long` and decoding it with lbits2d.

`def readutf(): String;`

Reads a string that has been encoded using a modified UTF-8 format. First, two bytes are read as unsigned integer. This value specifies are number of bytes that represent a string. The following bytes then decoded using ba2utf.

`def OStream.writebool(b: Bool);`

Writes `Bool` value to the output. Writes 1 if argument is `true` and 0 if argument is `false`.

`def OStream.writebyte(b: Int);`

Writes byte to the output. The byte is represented by low 8 bits of given value.

`def OStream.writeshort(s: Int);`

Writes short value (or character) to the output. The value is represented by low 16 bits of given value and is written as two bytes.

`def OStream.writeint(i: Int);`

Writes `Int` value to the output. Four bytes are written.

`def OStream.writelong(l: Long);`

Writes `Long` value to the output. Eight bytes are written.

`def OStream.writefloat(f: Float);`

Writes `Float` value to the output. It does so by converting given value to `Int` using f2ibits and writing resulting four bytes.

`def OStream.writedouble(d: Double);`

Writes `Double` value to the output. It does so by converting given value to `Long` using d2lbits and writing resulting eight bytes.

`def OStream.writeutf(str: String);`

Writes string in modified UTF-8 representation to the output. String is converted to byte sequence using String.utfbytes method. First, length of the sequence is written as unsigned short value. Then the sequence itself is written.

`def writebool(b: Bool);`

Writes a `Bool` value to stdout. Writes 1 if argument is `true` and 0 if argument is `false`.

`def writebyte(b: Int);`

Writes byte to stdout. The byte is represented by low 8 bits of given value.

`def writeshort(s: Int);`

Writes short value (or character) to stdout. The value is represented by low 16 bits of given value and is written as two bytes.

`def writeint(i: Int);`

Writes `Int` value to stdout. Four bytes are written.

`def writelong(l: Long);`

Writes `Long` value to stdout. Eight bytes are written.

`def writefloat(f: Float);`

Writes `Float` value to stdout. It does so by converting given value to `Int` using f2ibits and writing resulting four bytes.

`def writedouble(d: Double);`

Writes `Double` value to stdout. It does so by converting given value to `Long` using d2lbits and writing resulting eight bytes.

`def writeutf(str: String);`

Writes string in modified UTF-8 representation to stdout. String is converted to byte sequence using String.utfbytes method. First, length of the sequence is written as unsigned short value. Then the sequence itself is written.

5 textio.eh — Text I/O streams.

`use "textio.eh"`

5.1 Description

This header provides wrappers for I/O streams, allowing to process text in different encodings. Reader is a text input stream and Writer is a text output stream. Their API copies that of `IStream` and `OStream` with the only difference that they work with characters, not bytes.

To create new reader simply wrap `IStream` with suitable reader function.

```
var r = utfreader(fopen_r("sometext.txt"))
var ch = r.read()         // reads a single character
var line = r.readline()   // reads a line of characters
while (line != null) {
  line = r.readline()
  ...
}
r.close()
```

Wrappers are provided for the following encodings:

- *ISO 8859-1* (Latin-1, ASCII) - latin1reader and latin1writer;
- *UTF-8* - utfreader and utfwriter;
- *UTF-16* (little ending) - utf16reader and utf16writer.

You can create a text stream for arbitrary encoding (even self made) using Reader.new and Writer.new providing encoding functions for them.

UTF-8 is the default encoding used in Alchemy OS. It is used, for example, in terminal.

5.2 Types

```
type Reader < Any;
```

Text input stream.

```
type Writer < Any;
```

Text output stream.

5.3 Functions

```
def Reader.new(in: IStream, dec: (IStream): Int): Reader;
```

Creates new reader using the specified input stream and decoding function. On each call the decoding function should return a single character read from the given input stream. If the end of the stream is reached, this function should return EOF.

```
def Reader.read(): Int;
```

Reads next character from this reader. If the end of the stream is reached then EOF is returned.

```
def Reader.readarray(buf: [Char], ofs: Int, len: Int): Int;
```

Reads *len* characters from the input stream and writes them to the character array *buf* starting at index *ofs*. The function can end up in reading less characters if the end of the stream is reached. Returns total number of actually read characters. If no characters can be read because stream has reached its end then EOF is returned.

```
def Reader.readstr(len: Int): String;
```

Reads specified number of characters and returns them as the string. The function can end up reading less characters if the end of the streas is reached. If no characters can be read then `null` is returned.

```
def Reader.readline(): String;
```

Reads a single line of characters from this reader. If the end of the stream is reached then `null` is returned.

```
def Reader.skip(n: Long): Long;
```

Skips over and discards *num* characters. The function may, for a variety of reasons, end up skipping over some smaller number of characters, possibly 0. Returns total number of actually skipped characters.

```
def Reader.close();
```

Closes this reader and frees associated resources.

```
def Writer.new(out: OStream, enc: (OStream, Int)): Writer;
```

Creates new writer using the specified output stream and encoding function. On each call the encoding function should write the bytes representing given character to the output stream.

`def Writer.write(ch: Int);`

Writes given character to the output.

`def Writer.writearray(buf: [Char], ofs: Int, len: Int);`

Writes *len* characters from given character array starting from index *ofs* to the output.

`def Writer.print(str: String);`

Writes given string to the output.

`def Writer.println(str: String);`

Writes given string to the output and terminates a line.

`def Writer.printf(fmt: String, args: [Any]);`

Writes formatted string to the output using given format string and arguments. See String.format description on how formatting is used.

`def Writer.flush();`

Forces any buffered data to be written immediately.

`def Writer.close();`

Closes this writer and frees associated resources.

`def utfreader(in: IStream): Reader;`

Wraps given stream in a reader that uses UTF-8 encoding.

`def utfwriter(out: OStream): Writer;`

Wraps given stream in a writer that uses UTF-8 encoding.

`def latin1reader(in: IStream): Reader;`

Wraps given stream in a reader that uses ISO 8859-1 encoding (also known as Latin-1).

`def latin1writer(out: OStream): Writer;`

Wraps given stream in a writer that uses ISO 8859-1 encoding (also known as Latin-1).

`def utf16reader(in: IStream): Reader;`

Wraps given stream in a reader that uses UTF-16 encoding (little ending).

`def utf16writer(out: OStream): Writer;`

Wraps given stream in a writer that uses UTF-16 encoding (little ending).

6 pipe.eh — FIFO byte channels.

```
use "pipe.eh"
```

6.1 Description

Pipes are byte-transferring channels with two ends. Data written to the pipe output may then be read from the pipe input. Pipes can be used to create communication channels between processes.

6.2 Types

```
type Pipe < StreamConnection;
```

A FIFO communication channel.

6.3 Functions

```
def Pipe.new(): Pipe;
```

Creates new pipe.

7 string.eh — String operations.

```
use "string.eh"
```

7.1 Description

String is a sequence of unicode characters. Every character is a number in range 0..65535. Strings are immutable, once created string cannot change. If you need changeable strings, use either StrBuf or [Char].

Starting from release 2.0, you can use square brackets `[]` to get individual character or substring:

```
var str = "abcdef"
// Use [at] to get character
var ch = str[2]
// Use [from:to] to get substring
println(str[1:3])  /* bc */
// If first argument omitted, then it is beginning of the string
println(str[:4])   /* abcd */
// If last argument omitted, then it is end of the string
println(str[3:])   /* def */
```

7.2 Functions

```
def Int.tobin(): String;
```

Returns string representing this `Int` as an unsigned binary number.

```
def Int.tooct(): String;
```

Returns string representing this `Int` as an unsigned octal number.

```
def Int.tohex(): String;
```

Returns string representing this `Int` as an unsigned hexadecimal number.

```
def Int.tobase(base: Int): String;
```

Returns string representing this `Int` as a signed number in the given base. Argument *base* should be in range 2..36.

```
def Long.tobase(base: Int): String;
```

Returns string representing this `Long` as a signed number in the given base. Argument *base* should be in range 2..36.

```
def String.toint(): Int;
```

Parses this string as a signed decimal integer. If string is not representing valid number, then `null` is returned.

```
def String.tointbase(base: Int): Int;
```

Parses this string as a signed integer in the given base. Argument @emph base should be in range 2..36.

```
def String.tolong(): Long;
```

Parses this string as a signed decimal long integer. If string is not representing valid number, then `null` is returned.

```
def String.tolongbase(base: Int): Long;
```

Parses this string as a signed long integer in the given base. Argument *base* should be in range 2..36.

```
def String.tofloat(): Float;
```

Parses this string as a single precision floating point number. If string is not representing valid number, then `null` is returned.

```
def String.todouble(): Double;
```

Parses this string as a double precision floating point number. If string is not representing valid number, then `null` is returned.

```
def String.indexof(ch: Char): Int;
```

Returns index of the first occurence of given character in the string. If string does not contain given character then -1 is returned.

```
def String.lindexof(ch: Char): Int;
```

Returns index of the last occurence of given character in the string. If string does not contain given character then -1 is returned.

```
def String.startswith(prefix: String, ofs: Int = 0): Bool;
```

Tests if this string starts with the specified prefix. If offset is given, prefix string is tested at the corresponding position of this string.

```
def String.endswith(suffix: String): Bool;
```

Tests if the string ends with the specified suffix.

```
def String.replace(oldch: Char, newch: Char): String;
```

Returns new string where all occurences of the old character are replaced by new character.

```
def String.find(sub: String): Int;
```

Returns index of the first occurence of given substring in the string. If string does not contain given substring then -1 is returned.

```
def String.ucase(): String;
```

Returns string with all characters converted to the upper case.

```
def String.lcase(): String;
```

Returns string with all characters converted to the lower case.

```
def String.concat(str: String): String;
```

Concatenates this string with another string. The same is done by operator '+'.

```
def String.cmp(str: String): Int;
```

Compares this string with another string lexicographically. Returns zero if two strings are equal, value less than zero if this string is less than argument and value greater than zero if this string is greater than argument.

```
def String.trim(): String;
```

Removes all whitespaces from the beginning and the end of the stream. This includes removing of `' '`, `'\t'`, `'\r'` and `'\n'`.

```
def String.split(ch: Char): [String];
```

Splits the string around given character. Examples:

```
"foo,and,baaz".split(',')  // gives array ["foo", "and", "baaz"]
"foo,and,baaz".split('a')  // gives array ["foo,", "nd,b", "", "z"]
```

```
def String.format(args: [Any]): String;
```

Returns formatted string using this string as format specification and given arguments. Format specifiers are substrings of the form `%n` where `n` is from 0 to 9. Each format specifier is substituted with corresponding value from array *args*. Specifier `%%` is substituted with percent character. For example:

```
"The price of %0 has decreased by %1%%".format(["cookies", 8])
// gives "The price of cookies has decreased by 8%"
```

```
def String.chars(): [Char];
```

Returns characters of this string as character array.

```
def String.utfbytes(): [Byte];
```

Encodes this string in modified UTF-8 format and returns it as byte array.

```
def String.hash(): Int;
```

Computes 32-bit hash of this string using default Java hashing algorithm.

```
def ca2str(ca: [Char]): String;
```

Creates new string with characters given from specified character array.

```
def ba2utf(ba: [Byte]): String;
```

Decodes byte sequence as string in modified UTF-8 format.

8 strbuf.eh — String buffers.

```
use "strbuf.eh"
```

8.1 Description

String buffer is a mutable sequence of characters. Many of `StrBuf` functions return the string buffer after operation allowing to write sequences like

```
strbuf.append("restart").delete(0,2).insert(4, "le")
```

String buffers are essentially effective when you need to construct new string. Compare:

```
// slow
var str = ""
for (var i=0, i<100, i+=1)
  str += i


// fast
var sb = new StrBuf()
for (var i=0, i<100, i+=1)
  sb.append(i)

var str = sb.tostr()
```

In the first cycle new string is created on each iteration, wasting memory and time on memory allocation. The second cycle is more efficient in both memory and time. You may still use string concatenation for simple cases though, compiler automatically turns long concatenations into `StrBuf` calls.

```
// source string
println("sin" + "(" + a + ") = " + sin(a))

// compiler turns into
println(new StrBuf().append("sin(").append(a)
  .append(") = ").append(sin(a)).tostr())
```

8.2 Types

```
type StrBuf < Any;
```

Mutable character sequence. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed.

8.3 Functions

```
def StrBuf.new(): StrBuf;
```

Creates new empty string buffer.

```
def StrBuf.ch(at: Int): Char;
```

Returns character at the specified position of this string buffer. You may also use square brackets to get character (`strbuf[at]`).

```
def StrBuf.append(a: Any): StrBuf;
```

Appends text representation of given value to the end of this string buffer. The string representation is the same as returned by Any.tostr. Note that character literals are `Int` numbers so for example calling `strbuf.append('!')` actually adds string `"33"` to the string buffer *strbuf*. To add single character use addch function.

```
def StrBuf.addch(ch: Char): StrBuf;
```

Adds given character to the end of the string buffer.

```
def StrBuf.insert(at: Int, a: Any): StrBuf;
```

Inserts string representation of given value at specified index of this string buffer. The string representation is the same as returned by Any.tostr.

```
def StrBuf.insch(at: Int, ch: Char): StrBuf;
```

Inserts given character at specified index of this string buffer.

```
def StrBuf.replace(from: Int, to: Int, by: String): StrBuf;
```

Replaces substring of this string buffer by the string representation of given value. The string representation is the same as returned by Any.tostr.

```
def StrBuf.setch(at: Int, ch: Char): StrBuf;
```

Replaces character at specified index of this string buffer with given one.

```
def StrBuf.delete(from: Int, to: Int): StrBuf;
```

Removes all characters in given range from this string buffer.

```
def StrBuf.delch(at: Int): StrBuf;
```

Removes character at specified index from this string buffer.

```
def StrBuf.len(): Int;
```

Returns current number of characters in this string buffer.

```
def StrBuf.chars(from: Int, to: Int, buf: [Char], ofs: Int);
```

Copies characters from given range of this string buffer to the character array *buf* starting at offset *ofs*.

9 list.eh — Ordered collection of elements.

```
use "list.eh"
```

9.1 Description

A `List` is an ordered collection of elements (also known as *sequence*). Like in array, elements of list are enumerated by integer index from 0 to len()-1, however size of list is not fixed and may increase if needed. Lists support the same syntax as arrays. This header also provides convenient functions to sort, map, filter and fold lists.

```
var list = new List()
list.addall(["tar", "gate"])
list[0] = "s" + list[0]
println(list)
```

9.2 Types

```
type List < Any;
```

Sequence of elements.

9.3 Functions

```
def List.new(): List;
```

Creates new empty list.

```
def List.len(): Int;
```

Returns the number of elements in this list.

```
def List.get(at: Int): Any;
```

Returns element at the specified position of this list.

```
def List.set(at: Int, val: Any);
```

Replaces the element at the specified position in this list with the given value.

```
def List.add(val: Any);
```

Adds new element to the end of this list. Length of list increases by one.

```
def List.addall(vals: Array);
```

Adds all elements from the specified array to the end of this list. Length of list increases by length of given array.

```
def List.insert(at: Int, val: Any);
```

Inserts element at the specified position of this list. Length of list increases by one.

```
def List.insertall(at: Int, vals: Array);
```

Inserts all elements from given array at the specified position of this list. Length of list increases by length of given array.

```
def List.remove(at: Int);
```

Removes element at the specified position of the list decreasing its length by one.

```
def List.clear();
```

Removes all elements from the list.

```
def List.range(from: Int, to: Int): List;
```

Creates new list containing elements in given range of this list.

```
def List.indexof(val: Any): Int;
```

Returns index of the first occurence of given value in this list. If list does not contain given value, returns `-1`.

```
def List.lindexof(val: Any): Int;
```

Returns index of the last occurence of given value in this list. If list does not contain given value, returns `-1`.

```
def List.filter(f: (Any): Bool): List;
```

Creates new list containing only elements from this list that match given condition.

```
def List.filterself(f: (Any): Bool);
```

Filters this list leaving only elements that match given condition.

```
def List.map(f: (Any): Any): List;
```

Creates new list with elements obtained by applying given mapping to elements of this list.

```
def List.mapself(f: (Any): Any);
```

Transforms elements of this list using given mapping.

```
def List.reduce(f: (Any, Any): Any): Any;
```

Returns value obtained by sequential applying of given mapping to the elements of this list. If this list has no elements (its size is 0), this function returns `null`. If the list contains only one value, that value is returned. Otherwise, function f is applied to the first two elements of the list. Then f is applied to the result and the third element and so on.

```
def List.sort(f: (Any, Any): Int): List;
```

Creates new list with elements from this list sorted by given criteria. The function given as argument should return 0 if its arguments are equal, value less then 0 if its first argument is less then second and value greater then 0 if its first argument is greater then second.

```
def List.sortself(f: (Any, Any): Int);
```

Sorts elements of this list using given criteria. The function given as argument should return 0 if its arguments are equal, value less then 0 if its first argument is less then second and value greater then 0 if its first argument is greater then second.

```
def List.reverse(): List;
```

Creates new list with elements from this list in reverse order.

```
def List.toarray(): [Any];
```

Allocates new array and fills it with elements of this list. If you need to create an array of the other type then `Any`, use List.copyinto.

```
def List.copyinto(from: Int, buf: Array, ofs: Int, len: Int);
```

Copies contents of the list into given array. Method copies *len* elements starting from index *from* to the array *buf* starting at the array index *ofs*.

```
def List.tostr(): String;
```

Returns text representation of this list. The returned string has the form

`"[element1, element2, ...]"`.

```
def List.eq(other: List): Bool;
```

Compares the list with another list for equality. Returns `true` if two lists has the same length and corresponding elements are equal.

10 dict.eh — Dictionaries.

```
use "dict.eh"
```

10.1 Description

Dictionary is a set of key-value pairs. All keys are unique within one dictionary. Every non-`null` value can be used as a key or as a value.

Dictionaries support syntax similar to that of arrays.

```
var dict = new Dict()
dict["Name"] = "John"
dict["Surname"] = "Doe"
```

10.2 Types

```
type Dict < Any;
```

A dictionary.

10.3 Functions

```
def Dict.new(): Dict;
```

Creates new empty dictionary.

```
def Dict.set(key: Any, val: Any);
```

Maps given key to specified value.

```
def Dict.get(key: Any): Any;
```

Returns value to which given key is mapped in the dictionary. Returns `null` if the key is not stored in dictionary.

```
def Dict.remove(key: Any);
```

Removes specified key and its corresponding value from the dictionary. Does nothing if dictionary does not contain given key.

```
def Dict.clear();
```

Removes all keys and values from this dictionary.

```
def Dict.keys(): [Any];
```

Returns all keys currently stored in the dictionary.

```
def Dict.size(): Int;
```

Returns number of pairs in this dictionary.

```
def Dict.tostr(): String;
```

Returns string representation for this dictionary. Returned string has the form

```
"{key1=value1, key2=value2, ...}".
```

11 math.eh — Math functions.

```
use "math.eh"
```

11.1 Description

Common mathematical and trigonometric functions and constants.

11.2 Constants

```
const PI = 3.141592653589793;
```

Value of π, the ratio of the circumference of a circle to its diameter.

```
const POS_INFTY = 1d/0d;
```

Special `Double` value ”positive infinity”.

```
const NEG_INFTY = -1d/0d;
```

Special `Double` value ”negative infinity”.

```
const NaN = 0d/0d;
```

Special `Double` value ”not a number”.

```
const FPOS_INFTY = 1f/0f;
```

Special `Float` value ”positive infinity”.

```
const FNEG_INFTY = -1f/0f;
```

Special `Float` value ”negative infinity”.

```
const FNaN = 0f/0f;
```

Special `Float` value ”not a number”.

11.3 Functions

```
def abs(val: Double): Double;
```

Returns the absolute value of an argument.

```
def sgn(val: Double): Int;
```

Returns the sign of an argument. Returns `0` if value is zero, `-1` if value less then zero and `+1` if value is greater then zero.

```
def deg2rad(val: Double): Double;
```

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

```
def rad2deg(val: Double): Double;
```

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.

```
def sin(val: Double): Double;
```

Returns the trigonometric sine of an angle.

```
def cos(val: Double): Double;
```

Returns the trigonometric cosine of an angle.

```
def tan(val: Double): Double;
```

Returns the trigonometric tangent of an angle.

```
def sqrt(val: Double): Double;
```

Returns the square root of an argument.

```
def ipow(val: Double, pow: Int): Double;
```

Returns a^n, the value of the first argument raised to the power of the second argument.

```
def exp(val: Double): Double;
```

Returns Euler's number e raised to the power of an argument.

```
def log(val: Double): Double;
```

Returns the natural logarithm of an argument.

`def asin(val: Double): Double;`

Returns the arc sine of a value. The returned angle is in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

`def acos(val: Double): Double;`

Returns the arc cosine of a value. The returned angle is in the range $[0, \pi]$.

`def atan(val: Double): Double;`

Returns the arc tangent of a value. The returned angle is in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

`def ibits2f(bits: Int): Float;`

Decodes `Float` value from the given bit representation. The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "single format" bit layout.

`def f2ibits(f: Float): Int;`

Encodes `Float` value to the bit representation according to the IEEE 754 floating-point "single format" bit layout.

`def lbits2d(bits: Long): Double;`

Decodes `Double` value from the given bit representation. The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "double format" bit layout.

`def d2lbits(d: Double): Long;`

Encodes `Double` value to the bit representation according to the IEEE 754 floating-point "double format" bit layout.

12 rnd.eh — Pseudorandom numbers.

`use "rnd.eh"`

12.1 Description

Pseudorandom number generators.

12.2 Functions

```
def rnd(n: Int): Int;
```

Returns a pseudorandom, uniformly distributed integer value in range from 0 (inclusive) through n (exclusive).

```
def rndint(): Int;
```

Returns a pseudorandom, uniformly distributed integer value.

```
def rndlong(): Long;
```

Returns a pseudorandom, uniformly distributed long value.

```
def rndfloat(): Float;
```

Returns a pseudorandom float value, uniformly distributed between `0.0f` and `1.0f`.

```
def rnddouble(): Double;
```

Returns a pseudorandom double value, uniformly distributed between `0.0` and `1.0`.

13 sys.eh — Various system functions

```
use "sys.eh"
```

13.1 Description

This header contains various functions which do not fit in other headers.

13.2 Functions

```
def getenv(key: String): String;
```

Returns value of environment variable with given name. If variable is not defined then empty string `""` is returned.

```
def setenv(key: String, val: String);
```

Sets new value to the environment variable with given name.

```
def sleep(millis: Int);
```

Causes process to sleep for the specified number of milliseconds.

```
def exec_wait(prog: String, args: [String]): Int;
```

Executes given command with given arguments as subprocess of this process. This function blocks until subprocess executes and returns exit code of that subprocess.

```
def exec(prog: String, args: [String]);
```

Executes given command with given arguments as subprocess of this process in separate thread. This function returns immediately after subprocess starts and its result are two processes running consequently.

```
def sys_property(key: String): String;
```

Returns system property for the specified key. Returns `null` if there is no system property for given key.

14 time.eh — Time and date.

```
use "time.eh"
```

14.1 Description

Time is represented by `Long` value of milliseconds passed since ”the epoch” (January 1, 1970, 00:00:00 GMT).

14.2 Functions

```
def systime(): Long;
```

Returns current system time.

```
def datestr(time: Long): String;
```

Converts time to a string. String has the form `"dow mon dd hh:mm:ss zzz yyyy"`, where:

- `dow` is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat).
- `mon` is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).

- `dd` is the day of the month (01 through 31), as two decimal digits.
- `hh` is the hour of the day (00 through 23), as two decimal digits.
- `mm` is the minute within the hour (00 through 59), as two decimal digits.
- `ss` is the second within the minute (00 through 59), as two decimal digits.
- `zzz` is the time zone (and may reflect daylight savings time). If time zone information is not available, then `zzz` is empty.
- `yyyy` is the year, as four decimal digits.

```
def year(time: Long): Int;
```

Returns year represented by this time.

```
def month(time: Long): Int;
```

Returns month represented by this time as value in range from 1 to 12.

```
def day(time: Long): Int;
```

Returns day of month represented by this time.

```
def dow(time: Long): Int;
```

Returns day of week represented by this time as value in range from 1 to 7. Value 1 corresponds to Sunday.

```
def hour(time: Long): Int;
```

Returns hour of day represented by this time as value in range from 0 to 23.

```
def minute(time: Long): Int;
```

Returns minute within hour represented by this time as value in range from 0 to 59.

```
def second(time: Long): Int;
```

Returns second within minute represented by this time as value in range from 0 to 59.

```
def millis(time: Long): Int;
```

Returns millisecond within second represented by this time as value in range from 0 to 999.

```
def timeof(year: Int, month: Int, day: Int, hour: Int,
        min: Int, sec: Int, millis: Int): Long;
```

Returns time constant corresponding to given date.

15 error.eh — Error handling.

```
use "error.eh"
```

15.1 Description

Errors are thrown in exceptional situations, when program cannot proceed normally. `Error` object contains full trace of function calls to find out where error happened. For example this program:

```
// example.e
use "io"

def div(a: Int, b: Int): Int = a / b

def main(args: [String]) {
  println(div(1, 0))
}
```

produces the following output

```
Division by zero
@div(example.e:4)
@main(example.e:7)
```

Here you can find, what function sequence in what source caused the error (to make sources appear in trace of function calls you need to compile with -g option).

If you want to not pass error but handle it in your code, use `try/catch` block.

```
def main(args: [String]) {
  try {
    println(div(1, 0))
  } catch (var err) {
    println("Oops")
  }
}
```

In the last example, an `err` variable contains `Error` value. You can work with this value using functions from this header. Finally, you may throw your own errors using error function. Error code defines type of the error. Error codes 100..199 are reserved for system error types.

15.2 Constants

```
const SUCCESS = 0;
```

Indicates that no error occured. This convenient constant may be returned by main() if application ends normally. You cannot create an Error with zero error code.

```
const FAIL = 1;
```

Indicates that program ends with error. This convenient constant may be returned by main() if application ends abnormally.

```
const ERR_SYSTEM = 100;
```

Indicates that serious error occured in Alchemy OS. System error means that process cannot proceed normally due to system failure. For example, there is not enough memory.

```
const ERR_NULL = 101;
```

Indicates that operation failed because its argument is `null`. Null value usually means that you forgot to assign value to a variable or structure field. Also, elements of new arrays are unset, so they return null. Some functions may return null intentionally.

```
const ERR_IO = 102;
```

Indicates that operation failed because of input/output error. Errors of this kind may be thrown by all functions that work with files and network.

```
const ERR_RANGE = 103;
```

Indicates that an application tries to request or assign value that is out of given range. For example, this error is thrown when program tries to access array element at index less than zero or greater than array length.

```
const ERR_NEG_ALEN = 104;
```

Indicates that an application tries to create an array with negative size.

```
const ERR_ILL_ARG = 105;
```

Indicates that function has received illegal or inappropriate argument.

```
const ERR_ILL_STATE = 106;
```

Indicates that function has been called at an illegal or inappropriate time. In other words, application is not in appropriate state for the requested operation.

```
const ERR_SECURITY = 107;
```

Indicates security violation.

```
const ERR_TYPE_MISMATCH = 108;
```

Indicates that application has tried to operate on a value of the wrong type.

```
const ERR_DIV_BY_ZERO = 109;
```

Indicates that application has tried to divide by zero. This error is thrown only by integer arithmetics. Floating point division successfully returns NaN.

```
const ERR_INTERRUPT = 110;
```

Indicates that process was interrupted.

```
const ERR_MEDIA = 111;
```

Indicates that media playback failed.

15.3 Functions

```
def Error.code(): Int;
```

Returns error code of this error.

```
def Error.msg(): String;
```

Returns detail message of this error.

```
def Error.tostr(): String;
```

Converts this error to a string containing error message and trace of function calls.

```
def error(code: Int = FAIL, msg: String = null);
```

Raises error with given error code and detail message.

16 process.eh — Work with program processes.

```
use "process.eh"
```

16.1 Description

This header allows you to run one program from another in a more complex way than exec and exec_wait do. It provides routines to work with program processes including process of the current program.

A process lifecycle consists of three steps.

1. A process created with new_process() is initially in PS_NEW state. In this state you can configure it setting numerous attributes, such as priority, input/output streams, current working directory.

2. When you call start or start_wait process enters PS_RUNNING state. Running process executes program until it is ended by itself or interrupted.

3. When program ends, process enters final state PS_ENDED. In this state you can examine exit code of the process, and check if it ended with error.

16.2 Constants

```
const PS_NEW = 0;
```

A state of process that has been created but not yet started.

```
const PS_RUNNING = 1;
```

A state of process that has been started and not yet died.

```
const PS_ENDED = 5;
```

A state of process that has been terminated.

```
const MIN_PRIORITY = 1;
```

The minimum priority that a process can have.

```
const NORM_PRIORITY = 5;
```

The default priority assigned to a process.

```
const MAX_PRIORITY = 10;
```

The maximum priority that a process can have.

16.3 Types

```
type Process < Any;
```

Instance of executing program.

16.4 Functions

```
def this_process(): Process;
```

Returns reference to the process of current program.

```
def new_process(): Process;
```

Creates new process.

```
def Process.get_state(): Int;
```

Returns state of the process. State is one of `PS_*` constants defined in this header.

```
def Process.getenv(key: String): String;
```

Returns value of environment variable of this process.

```
def Process.setenv(key: String, value: String);
```

Sets value to the environment variable of this process.

```
def Process.get_in(): IStream;
```

Returns default input stream associated with the process.

```
def Process.get_out(): OStream;
```

Returns default output stream associated with the process.

```
def Process.get_err(): OStream;
```

Returns default error stream associated with the process.

```
def Process.set_in(in: IStream);
```

Sets new input stream for the process.

```
def Process.set_out(out: OStream);
```

Sets new output stream for the process.

`def Process.set_err(err: OStream);`

Sets new error stream for the process.

`def Process.get_cwd(): String;`

Returns current working directory of the process.

`def Process.set_cwd(dir: String);`

Sets current working directory of the process.

`def Process.start(prog: String, args: [String]);`

Starts specified program in this process. This function can be called only in PS_NEW state. After calling this function process enters PS_RUNNING state.

`def Process.start_wait(prog: String, args: [String]): Int;`

Starts specified program in this process and waits for it to die. This function can be called only when process is in PS_NEW state. Current process blocks until running process terminates.

`def Process.get_priority(): Int;`

Returns current priority of the process.

`def Process.set_priority(value: Int);`

Sets new priority to the process.

`def Process.get_name(): String;`

Returns name of the program this process executes. Returns `null` if the process is in PS_NEW state.

`def Process.interrupt();`

Interrupts the process. This function can be called only when process is in PS_RUNNING state. If the process is blocked waiting for some external operation, that operation is interrupted and process receives error ERR_INTERRUPT. If process is running normally, this function does nothing.

`def Process.get_error(): Error;`

If process ended with error, returns corresponding error object.

`def Process.get_exitcode(): Int;`

Returns exit code of the process.

17 dl.eh — Dynamic loading support.

```
use "dl.eh"
```

17.1 Description

This header provides support to open libraries and load functions from them at runtime. The common case for this is writing plugins.

```
var lib = loadlibrary("/lib/libcore.3.so")
var fun = cast ( (Double):Double ) lib.getfunc("sin")
println(fun(PI))    // outputs 0.0
```

17.2 Types

```
type Library < Any;
```

Represents instance of dynamically loaded library.

17.3 Functions

```
def loadlibrary(name: String): Library;
```

Loads and returns function library. If given name contains slash characters then it is used as path to the library. If not, then library named `libname.so` is searched in directories listed in environment variable `LIBPATH` (which is "/lib" by default). If function fails to load library it returns `null`.

```
def buildlibrary(input: IStream): Library;
```

Reads and constructs Ether library from the input stream.

```
def Library.getfunc(signature: String): Function;
```

Loads function with given signature from a library. If the library does not have a function with given signature then `null` is returned. Since type of the function is not known at compile time, you should explicitly cast function to appropriate type to use it.

18 connection.eh — Generic connection framework.

```
use "connection.eh"
```

18.1 Description

Connections are used by platform to connect to different sources of data. Usually you don't need to include this header as it is automatically pulled in by headers that define particular connections (such as HTTP connection or pipe connection).

18.2 Types

```
type Connection < Any;
```

A connection to system data source.

```
type StreamConnection < Connection;
```

A connection that has underlying input and output streams.

18.3 Functions

```
def Connection.close();
```

Closes the connection. Streams derived from the connection may be open when method is called. Any open streams will cause the connection to be held open until they themselves are closed. In this latter case access to the opened streams is permitted, but access to the connection is not.

```
def StreamConnection.open_input(): IStream;
```

Opens and returns input stream for connection.

```
def StreamConnection.open_output(): OStream;
```

Opens and returns output stream for connection.

19 func.eh — Functional transformations.

```
use "func.eh"
```

19.1 Description

This header defines functional transformations.

19.1.1 Partial function application

The curry function returns function with the first argument already applied. The type transformation is the following:

```
(type1,type2,...,typeN):rettype    =>    (type2,...,typeN):rettype
(type1,type2,...,typeN)            =>    (type2,...,typeN)
```

For example:

```
use "io"

def sum(a: Int, b: Int): Int = a + b

def main(args: [String]) {
  // the next function will be (Int):Int
  // and it will act as adding 3 to the argument
  var plus3 = sum.curry(3)
  println( plus3(2) )  // => 5
  // the function sum.curry itself is a "curried" function,
  // i.e. it is a chain of functions (Int):(Int):Int
  var chainsum = sum.curry
  println( chainsum(1)(8) )  // => 9
}
```

19.2 Functions

```
def Function.curry(arg: Any): Function;
```

Applies first argument to the function returning function with less arguments. The function being transformed must accept at least one argument. Actual returning type is calculated from the function being curried.