AScript ライブラリリファレンス

Updated: February 25, 2011

目次

1. この文書について

スクリプト言語 AScript の本体や標準添付のモジュールで定義されている関数やクラスの仕様について説明します。内容は AScript v0.01 の実装に基づきます（一部 v0.02 の実装内容を含みます)。

2. 組込み関数

2.1. インスタンス生成

```
audio(format:symbol, len:number, channels:number => 1)

binary(buff*)

codec(encoding:string, process_eol:boolean => 0)

color(elem?, green:number => 0, blue:number => 0, alpha:number => 255):map

dict(elem[]?) {block?}

dim(n+:number) {block?}

image(args+):map

lambda(`args*) {block}

list(iter+:iterator)

matrix(nrows:number, ncols:number, value?)

object()

pack(format:string, value*):map


palette(type:symbol)

semaphore()

set(iter+:iterator):[and,or,xor]

struct(`args+):[loose] {block?}

xlist(iter+:iterator)

xset(iter+:iterator):[and,or,xor]
```

2.2. 制御構文

2.2.1. 繰り返し

```
cross (`expr+) {block}
```

```
for (`expr+) {block}
```

```
repeat (n?:number) {block}
```

```
while (`cond) {block}
```

2.2.2. 条件分岐

```
if (`cond) {block}
```

```
elsif (`cond) {block}
```

```
else() {block}
```

2.2.3. 例外処理

```
try() {block}
```

```
except(errors*:error) {block}
```

```
raise(error:error, msg:string => "error", value?)
```

2.2.4. switch 文

```
switch() {block}
```

```
case(`cond) {block}
```

```
default() {block}
```

2.3. データ変換

```
chr(num:number):map
```

```
int(value):map
```

```
ord(str:string):map
```

```
tonumber(value):map:[nil,zero,raise,strict]
```

```
tostring(value):map
```

```
tosymbol(str:string):map
```

```
hex(num:number, digits?:number):map:[upper]
```

2.4. クラス操作

```
class(superclass?:function):[static] {block?}
```

```
classref(type:expr):map
```

2.5. ストリーム操作

```
copy(src:stream, dst:stream, bytesunit:number => 65536):map:void {block?}
open(name:string, mode:string => "r", encoding:string => "utf-8"):map {block?}
readlines(stream?:stream):[chop] {block?}
```

2.6. 変数スコープ操作

```
dir(obj?)
extern(`syms+)
local(`syms+)
locals(module?:module)
outers()
typename(`value)
undef(`value+):[raise]
```

2.7. イテレータ生成

```
fill(n:number, value?) {block?}
interval(a:number, b:number, samples:number):map:[open,open_l,open_r] {block?}
iterator(value+)
range(num:number, num_end?:number, step?:number):map {block?}
```

2.8. フォーマット変換

```
format(format:string, values*):map
zip(values+) {block?}
```

2.9. モジュール

```
import(`module, `alias?):[overwrite] {block?}
in(n, m)
module() {block}
```

2.10. データ型チェック

関数	説明

isbinary(value)	
isboolean(value)	
isclass(value)	
iscomplex(value)	
isdatetime(value)	
isdict(value)	
isenvironment(value)	
iserror(value)	
isexpr(value)	
isfunction(value)	
isiterator(value)	
islist(value)	
ismatrix(value)	
ismodule(value)	
isnumber(value)	
issemaphore(value)	
isstring(value)	
issymbol(value)	
istimedelta(value)	
isuri(value)	

```
isdefined(`symbol)
```

```
isinstance(value, type:expr):map
```

```
istype(value, type:expr):map
```

2.11. 演算・統計

```
choose(index:number, values+):map
```

```
chooseif(flag:boolean, value1, value2):map
```

```
max(values+):map
```

```
min(values+):map
```

```
mod(n, m):map
```

2.12. テキスト表示

```
print(value*):map:void
```

```
printf(format:string, values*):map:void
```

```
println(value*):map:void
```

2.13. スクリプト評価

```
eval(expr:expr):map
```

```
parse(code:string):map
```

```
parsestream(stream:stream):map
```

2.14. 乱数

```
rand(range?:number)
```

```
rands(num?:number, range?:number) {block?}
```

2.15. その他

```
help(func:function):map:void
```

3. 組込みクラス

3.1. binary クラス

3.1.1. インスタンスの生成

- コード中にバイナリリテラルを記述すると、binary インスタンスの生成になります。
- 関数 pack でバイナリ列のフォーマット指定をし、数値や文字列を埋め込んだインスタンスを生成することができます。

3.1.2. メソッドリファレンス

```
binary#add(buff+:binary)
```

```
binary#decode(codec:codec)
```

```
binary#dump():void:[upper]
```

```
binary#each() {block?}
```

```
binary#len()

binary#pointer(offset:number => 0)

binary#store(offset:number, buff+:binary)

binary#stream()

binary#unpack(format:string, offset:number => 0)

binary#unpacks(format:string, offset:number => 0, cnt?:number)
```

3.2. codec クラス

3.2.1. インスタンスの生成

- 関数 codec でインスタンスを生成します。

3.2.2. メソッドリファレンス

```
codec#decode(buff:binary):map

codec#encode(string:string):map
```

3.3. color クラス

3.3.1. インスタンスの生成

- 関数 color でインスタンスを生成します。

3.3.2. メンバ変数

変数名	読み書き	説明
red	R/W	赤要素を 0 から 255 までの数値で表します
green	R/W	緑要素を 0 から 255 までの数値で表します
blue	R/W	青要素を 0 から 255 までの数値で表します
alpha	R/W	アルファ要素を 0 から 255 までの数値で表します
gray	R	グレー値を取得します

3.3.3. メソッドリファレンス

```
color#html()
```

```
color#tolist(alpha:boolean => false)
```

3.4. datetime クラス

3.4.1. インスタンスの生成

- 関数 `time.datetime` で、年月日および時刻を指定したインスタンスを生成します。
- 関数 `time.time` で、時刻を指定したインスタンスを生成します。
- 関数 `time.today` で、今日の日付が入ったインスタンスを生成します。
- 関数 `time.now` で、現在の年月日および時刻が入ったインスタンスを生成します。

3.4.2. メソッドリファレンス

```
datetime#clrtzoff():reduce
```

```
datetime#format(format)
```

```
datetime#settzoff(mins:number):reduce
```

```
datetime#utc()
```

3.5. dict クラス

3.5.1. インスタンスの生成

- 辞書宣言 `%{…}` でインスタンスを生成します。
- 関数 dict でインスタンスを生成します。

3.5.2. メソッドリファレンス

```
dict#clear()
```

```
dict#erase(key):map
```

```
dict#get(key, default?:nomap):map:[raise]
```

```
dict#gets(key, default?):map:[raise]

dict#haskey(key):map

dict#items() {block?}

dict#keys() {block?}

dict#len()

dict#set(key, value:nomap):map:void

dict#setdefault(key, default)

dict#sets(key, value):map:void

dict#store(elems?):reduce {block?}

dict#values() {block?}
```

3.6. environment クラス

3.6.1. インスタンスの生成

3.6.2. メソッドリファレンス

3.7. error クラス

3.7.1. インスタンスの生成

3.7.2. メソッドリファレンス

3.8. expr クラス

3.8.1. インスタンスの生成

3.8.2. メソッドリファレンス

3.9. function クラス

3.9.1. インスタンスの生成

- 関数を定義すると、`function` インスタンスが生成されます。
- 関数 `lambda` でインスタンスを生成します。

3.9.2. メソッドリファレンス

```
function#argsymbols()
```

```
function#expr()
```

```
function#fullname()
```

```
function#help(help?:string)
```

```
function#name()
```

```
function#setsymbol(symbol:symbol):reduce
```

3.10. image クラス

3.10.1. インスタンスの生成

- 関数 `image` でインスタンスを生成します。

- イメージ操作に関わるモジュールに、image インスタンスを生成するものがあります。詳細は各モジュールの仕様を参照ください。

3.11. メソッドリファレンス

```
image#allocbuff(width:number, height:number, color?:number):void

image#crop(x:number, y:number, width?:number, height?:number):map

image#delpalette():reduce

image#each(x?:number, y?:number, width?:number, height?:number,
   scandir?:symbol) {block?}

image#extract(x:number, y:number, width:number, height:number, element:symbol,
   dst):void

image#fill(color:color):void

image#fillrect(x:number, y:number, width:number, height:number,
   color:color):map:void

image#flip(orient:symbol):map

image#getpixel(x:number, y:number):map

image#paste(x:number, y:number, src:image, width?:number, height?:number,
   xoffset:number => 0, yoffset:number => 0, alpha:number => 255):map:reduce

image#putpixel(x:number, y:number, color:color):map:void

image#read(stream:stream, imgtype?:string):map:reduce

image#reducecolor(palette?:palette)
```

```
image#resize(width?:number, height?:number):map:[box]

image#rotate(rotate:number, background?:color):map

image#setalpha(colorKey:color, alphaKey:number => 0, alphaRest:number =>
    255):reduce

image#size()

image#store(x:number, y:number, width:number, height:number, element:symbol,
    src):void

image#thumbnail(width?:number, height?:number):map:[box]

image#write(stream:stream, imgtype?:string):map:reduce
```

3.12. iterator クラス

3.12.1. インスタンスの生成

- 多くの生成方法があります。「AScript ランゲージリファレンス」のイテレータの章を参照ください。

3.12.2. メソッドリファレンス

イテレータが実装するメソッドは、リストのメソッドと大部分が共通しています。共通しているメソッドはリストのリファレンスに掲載していますので、そちらを参照ください。この項は、イテレータ特有のメソッドを示します。

```
iterator#delay(delay:number) {block?}

iterator#isinfinite()

iterator#next()
```

3.13. list クラス

3.13.1. インスタンスの生成

- リスト宣言 `[…]` や `@{…}` でインスタンスを生成します。

- 関数 `list` でインスタンスを生成します。

3.13.2. メソッドリファレンス

`list#add(elem+):reduce`

`list#align(n:number, value?):map {block?} / iterator#align(n:number, value?) {block?}`

`list#and() / iterator#and()`

`list#append(elem+):reduce`

`list#average() / iterator#average()`

要素から平均値を算出し、結果を返します。

`list#clear():reduce`

要素をすべてとりのぞき、空のリストにします。これは破壊的メソッドです。

`list#combination(n:number) {block?}`

`list#count(criteria?) / iterator#count(criteria?)`

条件に合致する要素の数を返します。条件 `criteria` には値または関数を指定します。
`criteria` を省略すると、要素中で真値と判断できるものの数を返します。
`criteria` に値を指定した場合、その値と要素を比較し、等しいと判断したものの数を数えます。
`criteria` に渡す関数は、引数を一つとり `boolean` 値を返すものを指定します。`list#count` メソッドは要素をひとつずつ関数に渡し、帰ってきた `true` の数を数えます。

`list#each() {block?} / iterator#each() {block?}`

リストの要素を順に走査するイテレータを返します。ブロック引数を指定すると、要素ごとにブロックの内容を評価します。ブロック引数は `|value, idx:number|` となります。`value` が要素値、`idx` が走査インデクスです。

`list#erase(idx*:number):reduce`

`list#filter(criteria) {block?} / iterator#filter(criteria) {block?}`

`list#first()`

`list#flat()`

`list#fold(n:number, nstep?:number):[iteritem] {block?} / iterator#fold(n:number):[iteritem] {block?}`

```
list#format(format:string):map / iterator#format(format:string) {block?}

list#get(index:number):map:flat

list#head(n:number):map {block?} / iterator#head(n:number):map {block?}

list#iscontain(value) / iterator#iscontain(value)

list#isempty()

list#join(sep:string => "") / iterator#join(sep?:string)

iterator#joinb()

list#last()

list#len() / iterator#len()

list#map(func:function) {block?} / iterator#map(func:function) {block?}

list#max():[index,last_index,indices]                                    /
   iterator#max():[index,last_index,indices]

list#min():[index,last_index,indices]                                    /
   iterator#min():[index,last_index,indices]

list#nilto(replace) / iterator#nilto(replace)

list#offset(n:number):map {block?} / iterator#offset(n:number) {block?}

list#or() / iterator#or()

list#pack(format:string) / iterator#pack(format:string) {block?}

list#permutation(n?:number) {block?}

list#pingpong(n?:number):[sticky,sticky_l,sticky_r]        {block?}        /
   iterator#pingpong(n?:number):[sticky,sticky_l,sticky_r] {block?}

iterator#print(stream?:stream)

list#printf(format:string):void       /       iterator#printf(format:string,
   stream?:stream)

iterator#println(stream?:stream)

list#rank(directive?):[stable] / iterator#rank(directive?) {block?}

list#reduce(accum) {block} / iterator#reduce(accum) {block}

list#replace(value, replace) / iterator#replace(value, replace)
```

`list#reverse() {block?} / iterator#reverse() {block?}`

`list#round(n?:number) {block?} / iterator#round(n?:number) {block?}`

`list#shift():[raise]`

`list#shuffle():reduce`

`list#since(criteria) {block?} / iterator#since(criteria) {block?}`

`list#skip(n:number):map {block?} / iterator#skip(n:number) {block?}`

`list#skipnil() {block?} / iterator#skipnil() {block?}`

`list#sort(directive?, keys[]?):[stable] / iterator#sort(directive?, keys[]?) {block?}`

要素の順番を並べ替えた結果をイテレータで返します。リストで結果を得る場合はアトリビュート:`list`を指定します。

並べ替えの順序はデフォルトで昇順ですが、引数 `directive` にシンボルまたは関数を指定することで順序を指示することができます。`directive` に、シンボル`ascend を指定すると昇順、シンボル`descend` を指定すると降順になります。

`directive` に関数を渡す場合、この関数は二つの引数をとり、`-1, 0, +1` のいずれかの整数値を返すものである必要があります。今、関数の一般式が `f(a, b)` であるとすると、以下のような値を返すようにします。

昇順: `a < b` のとき `-1`、`a == b` のとき `0`、`a > b` のとき `+1`

降順: `a > b` のとき `+1`、`a == b` のとき `0`、`a < b` のとき `+1`

`sort` メソッドは、デフォルトではリストの要素そのものの大小で並び替えを行いますが、引数 `keys` にリストを渡すと、これをキーとしてソート処理をします。`keys` の要素数はリストの要素数と同じでなければいけません。

アトリビュート:`stable` をつけると、ステーブルソートになります。大小比較が等しい要素が複数あったとき、それらの順序がソート前と同じである保障が得られます。

`list#stddev() / iterator#stddev()`

要素から標準偏差を算出し、結果を返します。

`list#sum() / iterator#sum()`

すべての要素を加算した結果を返します。

`list#tail(n:number):map {block?} / iterator#tail(n:number) {block?}`

`list#variance() / iterator#variance()`

要素から分散値を算出し、結果を返します。

```
list#while (criteria) {block?}
```

3.14. matrix クラス

3.14.1. インスタンスの生成

- マトリクス宣言 `@@{…}` でインスタンスを生成します。
- 関数 `matrix` でインスタンスを生成します。

3.14.2. メソッドリファレンス

```
matrix#col(col:number):map
```

```
matrix#colsize()
```

```
matrix#each():[transpose]
```

```
matrix#eachcol()
```

```
matrix#eachrow()
```

```
matrix#inverse()
```

```
matrix#issquare()
```

```
matrix#row(row:number):map
```

```
matrix#rowsize()
```

```
matrix#set(value)
```

```
matrix#setcol(col:number, value)
```

```
matrix#setrow(row:number, value)
```

```
matrix#submat(row:number, col:number, nrows:number, ncols:number):map
```

```
matrix#tolist():[flat,transpose]
```

```
matrix#transpose()
```

3.15. palette クラス

3.15.1. インスタンスの生成

- 関数 `palette` でインスタンスを生成します。

3.15.2. メソッドリファレンス

```
palette#each() {block?}
```

```
palette#nearest(color:color):map:[index]
```

```
palette#shrink():reduce
```

```
palette#updateby(image_or_palette):reduce:[shrink]
```

3.16. semaphore クラス

3.16.1. インスタンスの生成

- 関数 `semaphore` でインスタンスを生成します。

3.16.2. メソッドリファレンス

```
semaphore#release()
```

```
semaphore#session() {block}
```

```
semaphore#wait()
```

3.17. stream クラス

3.17.1. インスタンスの生成

- 関数 open でインスタンスを生成します。

3.17.2. メソッドリファレンス

```
stream#close()

stream#compare(stream:stream):map

stream#dosmode(dos_flag:boolean):reduce

stream#peek(len?:number)

stream#print(values*):map:void

stream#printf(format:string, values*):map:void

stream#println(values*):map:void

stream#read(len?:number)

stream#readline():[chop]

stream#readlines(nlines?:number):[chop] {block?}

stream#readtext()

stream#readto(stream:stream, bytesunit:number => 65536):map:reduce {block?}

stream#seek(offset:number, origin?:symbol):reduce

stream#setencoding(encoding:string, dos_flag?:boolean)
```

```
stream#tell()

stream#write(buff:binary):reduce

stream#writefrom(stream:stream, bytesunit:number => 65536):map:reduce {block?}
```

3.18. string クラス

3.18.1. インスタンスの生成

- コード中に文字列リテラルを記述すると、string インスタンスの生成になります。

3.18.2. メソッドリファレンス

```
string#align(len:number, padding:string => " "):map:[center,left,right]

string#capitalize()

string#each():map:[utf32] {block?}

string#eachline(nlines?:number):[chop] {block?}

string#encode(codec:codec)

string#endswith(suffix:string, endpos?:number):map:[icase]

string#escapehtml()

string#find(sub:string, pos:number => 0):map:[rev,icase]

string#format(values*):map

string#isempty()
```

```
string#left(len?:number):map

string#len()

string#lower()

string#mid(pos:number => 0, len?:number):map

string#print(stream?:stream):void

string#println(stream?:stream):void

string#replace(sub:string, replace:string, count?:number):map:[icase]

string#right(len?:number):map

string#split(sep?:string, count?:number):[icase] {block?}

string#startswith(prefix:string, pos:number => 0):map:[icase]

string#strip():[both,left,right]

string#unescapehtml()

string#upper()
```

3.19. timedelta クラス

3.19.1. インスタンスの生成

- 関数 `time.delta` でインスタンスを生成します。
- `datetime` インスタンス同士を引き算した結果は `timedelta` インスタンスです。

3.19.2. メソッドリファレンス

4. sys モジュール

AScript 本体の動作モードを変えたり、実行状態を得たりするモジュールです。

4.1. 関数

`sys.echo(flag:boolean)`

対話モードのとき、結果をエコーバックするか否かを設定します。`flag` に `true` を指定するとエコーバックが有効になります。

`sys.exit(status?:number)`

プログラムを終了します。`status` で終了コードを指定します。省略すると 0 になります。

`sys.listcodec()`

利用可能な文字コーデックの名前の一覧をリストで返します。

4.2. 変数

sys モジュールが定義・参照している変数は以下のとおりです。

変数	型	内容
`ps1`	string	対話モードで、インデントがかかっていないときのプロンプト
`ps2`	string	対話モードで、インデントがかかっている間のプロンプト
`stdin`	stream	標準入力に使うストリーム
`stdout`	stream	標準出力に使うストリーム
`stderr`	stream	標準エラー出力に使うストリーム
`path`	list	モジュールのサーチパスを記述したリスト
`version`	string	AScript のバージョン番号
`build`	symbol	AScript をビルドした環境をシンボルで表す。`gcc: Linux 環境の gcc, `bcc: Windows 環境の Borland C、`msc: Windows 環境の Visual Studio
`executable`	string	AScript 実行ファイルのフルパス名
`datadir`	string	AScript のデータディレクトリのフルパス名
`libdir`	string	AScript のライブラリディレクトリのフルパス名
`argv`	list	引数リスト。argv[0] にはスクリプトの名前がフルパスで格納される

5. fs モジュール

ファイルシステムの操作をするモジュールです。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- open 関数でファイルシステム上のファイルをオープンできるようになります。
- ストリームを受け取る引数に、ファイルシステム上のファイルパス名を指定できるようになります。
- path.dir, path.walk, path.glob 関数で、ファイルシステム上のディレクトリパスをサーチできるようになります。

5.1. 関数

6. os モジュール

OS の操作をまとめたモジュールです。

7. path モジュール

パス操作をまとめたモジュールです。

7.1. 関数

8. math モジュール

数学演算処理をまとめたモジュールです。

8.1. 関数

9. string モジュール

文字列処理をまとめたモジュールです。」

9.1. 関数

10. time モジュール

時刻操作をまとめたモジュールです。

10.1. 関数

11. hash モジュール

11.1. 関数

12. ネットワーク – net.http モジュール

HTTP プロトコルのサーバとクライアント処理を提供するモジュールです。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- open 関数で HTTP プロトコルを通じたファイルをオープンできるようになります。
- ストリームを受け取る引数に、HTTP のファイルパス名を指定できるようになります。

12.1. 関数

13. イメージファイル

image クラスに image#read()および image#write()
image#XXXread()および image#XXXwrite()

bmp, gif, png, jpeg, msicon, ppm, tiff

14. 二次元イメージ描画 – cairo モジュール

Cairo のオフィシャルサイトは http://cairographics.org/ です。

14.1. 命名規則

コンテキストは cairo.context クラスのインスタンスで表されます。コンテキストを操作する関数は、"cairo_" をとった名前のメソッドになります。例えば、cairo_set_source_rgb(cr, r, g, b) は cairo.context#set_source_rgb(r, g, b) となります。

パターンは cairo.pattern クラスのインスタンスで表されます。パターンを操作する関数は、"cairo_pattern_" をとった名前のメソッドになります。例えば、cairo_pattern_set_filter(pattern, filter) は pattern.set_filter(filter) となります。

グリフは cairo.glyph クラスのインスタンスで表されます。グリフを操作する関数は、"cairo_glyph_" とった名前のメソッドになります。例えば、cairo_glyph_

パスは cairo.path クラスのインスタンスで表されます。パスを操作する関数は、"cairo_path_" とった名前のメソッドになります。例えば、cairo_path_

サーフェースは cairo.surface クラスのインスタンスで表されます。サーフェースを操作する関数は、"cairo_surface_" とった名前のメソッドになります。例えば、cairo_surface_

デバイスは cairo.device クラスのインスタンスで表されます。デバイスを操作する関数は、

"cairo_device_" をとった名前のメソッドになります。例えば、cairo_device_

フォントオプションは cairo.font_options クラスのインスタンスで表されます。フォントオプションを操作する関数は、"cairo_font_options_" をとった名前のメソッドになります。例えば、cairo_font_options_

矩形は cairo.rectangle クラスのインスタンスで表されます。矩形を操作する関数は、"cairo_rectangle_" をとった名前のメソッドになります。例えば、cairo_rectangle_

テキストエクステントは cairo.text_extents クラスのインスタンスで表されます。テキストエキステントを操作する関数は、"cairo_text_extents_"をとった名前のメソッドになります。例えば、cairo_text_extents_

フォントエキステントは cairo.font_extents クラスのインスタンスで表されます。フォントエキステントを操作する関数は、"cairo_font_extents_" をとった名前のメソッドになります。例えば、cairo_font_extents_

マクロ変数は、その変数が属するグループに共通してつけられるプレフィックスをとりのぞき、大文字をすべて小文字にした名前のシンボルで表されます。例えば、CAIRO_LINE_CAP_BUTT や CAIRO_LINE_CAP_ROUND は、それぞれシンボル `butt および `round で表されます。

14.2. リファレンス

15. canvas モジュール

16. フォント描画 – freetype モジュール

17. 三次元描画 – opengl および glu モジュール

OpenGL は…

オフィシャルサイトは http://www.opengl.org/ です。

17.1. 命名規則

18. 高速画面描画 – sdl モジュール

SDL は…

オフィシャルサイトは http://www.libsdl.org/ です。

18.1. 命名規則

18.2. sdl.Cursor クラス

```
Cursor#FreeCursor():void
```

18.3. sdl.Timer クラス

```
Timer#RemoveTimer()
```

18.4. sdl.PixelFormat クラス

```
PixelFormat#GetRGB(pixel:number)
PixelFormat#GetRGBA(pixel:number)
PixelFormat#MapRGB(r:number, g:number, b:number)
PixelFormat#MapRGBA(r:number, g:number, b:number, a:number)
```

18.5. sdl.PixelFormat クラス

```
Surface#ConvertSurface(fmt:PixelFormat, flag:number) {block?}
Surface#DisplayFormat() {block?}
Surface#DisplayFormatAlpha() {block?}
Surface#FillRect(rect:Rect, color:Color):map:void
Surface#Flip()
Surface#GetClipRect()
Surface#LockSurface()
Surface#PutSurface(src:Surface, x:number => 0, y:number => 0):map
Surface#SaveBMP(file:string):void
Surface#SetAlpha(flag:number, alpha:number)
Surface#SetClipRect(rect:Rect):map:void
Surface#SetColorKey(flag:number, key:number)
Surface#SetColors(colors[]:Color, firstcolor:number => 0)
Surface#SetPalette(flags:number, colors[]:Color, firstcolor:number => 0)
Surface#UnlockSurface():void
Surface#UpdateRect(x:number => 0, y:number => 0, w:number => 0, h:number =>
0):void
Surface#UpdateRects(rects[]:Rect):void
```

18.6. sdl.Overlay クラス

```
Overlay#DisplayYUVOverlay(dstrect:Rect)
Overlay#LockYUVOverlay()
Overlay#UnlockYUVOverlay():void
```

18.7. sdl.Joystick クラス

```
Joystick#JoystickClose():void
Joystick#JoystickGetAxis(axis:number)
Joystick#JoystickGetBall(ball:number)
Joystick#JoystickGetButton(button:number)
Joystick#JoystickGetHat(hat:number)
Joystick#JoystickIndex()
Joystick#JoystickNumAxes()
```

```
Joystick#JoystickNumBalls()
Joystick#JoystickNumButtons()
Joystick#JoystickNumHats()
```

18.8. sdl.AudioSpec クラス

```
AudioSpec#MixAudio(src:AudioSpec, volume:number)
```

18.9. sdl.AudioCVT クラス

```
AudioCVT#ConvertAudio()
```

18.10. sdl.CD クラス

```
CD#CDClose():void
CD#CDEject()
CD#CDPause()
CD#CDPlay(start:number, length:number)
CD#CDPlayTracks(start_track:number, start_frame:number, ntracks:number,
nframes:number)
CD#CDResume()
CD#CDStatus()
CD#CDStop()
CD#GetTrack(n:number):void
```

19. グラフィカルユーザインターフェース – tcl および tk モジュール

Tcl/Tk のオフィシャルサイトは http://www.tcl.tk/ です。

19.1. モジュール構成

モジュールは、Tcl インタープリタとのインターフェースを提供するモジュール `tcl.azd` と、そのモジュールを使って Tk の機能を実現するモジュール `tk.az` からなります。Tcl/Tk には多くの関数がありますが、`tcl.azd` や `tk.az` の中でそれらの関数を個別に実装しているわけではありません。ユーザが呼び出したメソッド名から動的にメソッドを生成して実行しています。

19.2. 命名規則

ウィジェットを生成する手続きは、親ウィジェットを表す変数のメソッドとして実装されます。
ウィジェットを生成する手続きは、最初の文字を大文字にした名前のメソッドになります。例えば、canvas は Canvas になります。
名前空間 ttk に属する手続きは、`"ttk$"` につづけて、最初の文字を大文字にした名前をつけたメソッドになります。例えば、`ttk::button` は `ttk$Button` になります。

19.3. イベントの扱い

Tcl/Tk は、イベントが発生すると関連付けられたプロシージャを評価します。この際、プロシージャ中にパーセント記号 "%" と一文字のアルファベットまたは記号が記述されていると、評価の前にその部分をイベントに付随するパラメータ値に置換します。例えば、マウスをクリックしたときに発生する ButtonPress イベントが発生すると、関連付けたプロシージャ中の %x や %y という部分をそれぞれマウスの x, y 座標数値に置換してから評価が行われます。置換文字列の一覧は http://www.tcl.tk/man/tcl8.5/TkCmd/bind.htm にまとめられています。

AScript はブロックでイベントハンドラを指定できますが、イベントのパラメータ値はブロック引数として受け取ります。このとき、ブロック引数の名前が、Tcl/Tk の置換文字列の名前に対応します。上の例を使えば、ブロック引数に x, y という名前の引数を指定すると、これらに x, y 座標値が入ります。一般の関数呼び出しの慣例では引数の位置によって受け渡しする値を指定しますが、Tcl/Tk のイベントハンドラでは名前で受け取る値が決まることに注意してください。

また、Tcl/Tk が置換する結果は文字列になるので、数値など他の方で受け取りたい場合はブロック引数に明示的に型指定をしてください。

ブロック引数では、Tcl/Tk が定義する一文字の識別子のほかに、可読性を高めるため以下の別名も受け付けられるようにしています。

引数名	Tcl/Tk
serial	#
above	a
numbuttons	b
count	c
detail	d
user_data	d
focus	f
height	h
widget_num	i
keycode	k
mode	m
override_redirect	o

引数名	Tcl/Tk
place	p
state	s
time	t
width	w
x	x
y	y
char	A
border_width	B
delta	D
send_event	E
keysym	K
keysym_num	N

引数名	Tcl/Tk
root	R
subwidget	S
type	T
widget	W
x_root	X
y_root	Y
action	d
index	i
wouldbe	P
current	s
edited	S
validate	V

この対応づけは、tk.bindSubstDict 変数に辞書として登録されており、必要に応じて追加することができます。例えば、%h で置換されるイベントパラメータを hoge という引数名で受け取りたい場合は、以下のようなコードを追加します。

```
tk.bindSubstDict[`hoge] = "h"
```

19.4. モジュール関数

```
tk.bell(args*, opts%):map
tk.bind(args*, opts%):map
```

```
tk.bindtags(args*, opts%):map
tk.tkerror(args*, opts%):map
tk.update()
tk.winfo$atom(args*, opts%):map
tk.winfo$atomname(args*, opts%):map
tk.winfo$containing(args*, opts%):map
tk.winfo$interps(args*, opts%):map
tk.winfo$pathname(args*, opts%):map
tk.tk$FocusFollowsMouse(args*, opts%):map
tk.tk$SetPalette(args*, opts%):map
tk.tk$bisque(args*, opts%):map
tk.tkwait$variable(args*, opts%):map
tk.tkwait$visibility(args*, opts%):map
tk.tkwait$window(args*, opts%):map
tk.mainloop()
tk.tclDump(flag:boolean => true)
```

19.5. image クラスへの追加メソッド

tk モジュールをインポートすると、image クラスに以下のメソッドが追加されます。

```
image#to_tk()
```

19.6. tk.Widget クラス

```
Widget#wm$aspect(minNumber?, minDenom?, maxNumber?, maxDenom?):map
```

Tcl コマンド: `wm aspect window ?minNumber mindenom maxNumber maxDenom`

```
Widget#wm$attributes(opts%):map
```

Tcl コマンド: `wm attributes window`

```
Widget#wm$client(name?):map
```

Tcl コマンド: `wm client window ?name?`

```
Widget#wm$colormapwindows(windowList?):map
```

Tcl コマンド:

```
Widget#wm$command(value?):map
```

Tcl コマンド:

```
Widget#wm$deiconify():map
```

Tcl コマンド:

```
Widget#wm$focusmodel(args*, opts%):map
```

Tclコマンド:

Widget#wm$forget():map

Tclコマンド:

Widget#wm$frame():map

Tclコマンド:

Widget#wm$geometry(newGeometry):map

Tclコマンド:

Widget#wm$grid(baseWidth?, baseHeight?, widthInc?, heightInc?):map

Tclコマンド:

Widget#wm$group(args*, opts%):map

Tclコマンド:

Widget#wm$iconbitmap(args*, opts%):map

Tclコマンド:

Widget#wm$iconify(args*, opts%):map

Tclコマンド:

Widget#wm$iconmask(args*, opts%):map

Tclコマンド:

Widget#wm$iconname(args*, opts%):map

Tclコマンド:

Widget#wm$iconphoto(args*, opts%):map

Tclコマンド:

Widget#wm$iconposition(args*, opts%):map

Tclコマンド:

Widget#wm$iconwindow(args*, opts%):map

Tclコマンド:

Widget#wm$manage(args*, opts%):map

Tclコマンド:

Widget#wm$maxsize(args*, opts%):map

Tclコマンド:

Widget#wm$minsize(args*, opts%):map

Tclコマンド:

Widget#wm$overrideredirect(args*, opts%):map

Tclコマンド:

Widget#wm$positionfrom(args*, opts%):map

Tclコマンド:

Widget#wm$protocol(args*, opts%):map

Tclコマンド:

Widget#wm$resizable(args*, opts%):map

Tclコマンド:

Widget#wm$sizefrom(args*, opts%):map

Tclコマンド:

Widget#wm$stackorder(args*, opts%):map

Tclコマンド:

Widget#wm$state(args*, opts%):map

Tclコマンド:

Widget#wm$title(args*, opts%):map

Tclコマンド:

Widget#wm$transient(args*, opts%):map

Tclコマンド:

Widget#wm$withdraw():map

Tclコマンド:

Widget#winfo$(args*, opts%):map

Tclコマンド:

Widget#grid$(args*, opts%):map

Tclコマンド:

Widget#place$(args*, opts%):map

Tclコマンド:

Widget#tk$bisque(args*, opts%):map

Tclコマンド:

Widget#tk$chooseColor(args*, opts%):map

Tclコマンド:

Widget#tk$chooseDirectory(args*, opts%):map

Tclコマンド:

Widget#tk$dialog(args*, opts%):map

Tclコマンド:

Widget#tk$focusFollowsMouse(args*, opts%):map

Tclコマンド:

Widget#tk$focusNext(args*, opts%):map

Tclコマンド:

Widget#tk$focusPrev(args*, opts%):map

Tclコマンド:

Widget#tk$getOpenFile(args*, opts%):map

Tclコマンド:

Widget#tk$getSaveFile(args*, opts%):map

Tclコマンド:

Widget#tk$menuSetFocus(args*, opts%):map

Tclコマンド:

Widget#tk$messageBox(args*, opts%):map

Tclコマンド:

Widget#tk$optionMenu(args*, opts%):map

Tclコマンド:

Widget#tk$popup(args*, opts%):map

Tclコマンド:

Widget#tk$setPalette(args*, opts%):map

Tclコマンド:

Widget#tk$textCopy(args*, opts%):map

Tclコマンド:

Widget#tk$textCut(args*, opts%):map

Tclコマンド:

Widget#tk$textPaste(args*, opts%):map

Tclコマンド:

19.6.1. ウィジェット生成メソッド

例えば、button ウィジェットを生成するメソッドの一般式は以下のようになります。

```
tk.Widget#Button(options%)
```

このメソッドは、Tcl コマンドとして"`button pathName %options%`" を実行します。`pathName` は新しく生成する `button` ウィジェットのパス名、`options` は `tk.Widget#Button` メソッドに辞書形式で渡した引数の内容が渡されます。

ウィジェット生成メソッドの戻り値は、新しく生成したウィジェットのパス名を持った `tk.Widget` インスタンスです。menu ウィジェットや text ウィジェットのように、ウィジェット特有のメソッドを持つものについては、`tk.Widget` クラスから派生したクラスのインスタンスを返します。

ウィジェット生成メソッドと、生成する Tk ウィジェットの対応表を以下にまとめます。

メソッド	生成する Tk ウィジェット	メソッドの戻り値
`tk.Widget#Button`	button	`tk.Widget`
`tk.Widget#Canvas`	canvas	`tk.Widget`
`tk.Widget#Checkbutton`	checkbutton	`tk.Widget`
`tk.Widget#Entry`	entry	`tk.Widget`
`tk.Widget#Frame`	frame	`tk.Widget`
`tk.Widget#Label`	label	`tk.Widget`
`tk.Widget#Labelframe`	labelframe	`tk.Widget`
`tk.Widget#Listbox`	listbox	`tk.Widget`
`tk.Widget#Menu`	menu	`tk.Menu`
`tk.Widget#Menubutton`	menubutton	`tk.Widget`
`tk.Widget#Message`	message	`tk.Widget`
`tk.Widget#Panedwindow`	panedwindow	`tk.Widget`
`tk.Widget#Radiobutton`	radiobutton	`tk.Widget`
`tk.Widget#Scrollbar`	scrollbar	`tk.Widget`
`tk.Widget#Spinbox`	spinbox	`tk.Widget`
`tk.Widget#Text`	text	`tk.Text`
`tk.Widget#Toplevel`	toplevel	`tk.Widget`
`tk.Widget#ttk$Button`	ttk::button	`tk.Widget`
`tk.Widget#ttk$Checkbutton`	ttk::checkbutton	`tk.Widget`
`tk.Widget#ttk$Combobox`	ttk::combobox	`tk.Widget`
`tk.Widget#ttk$Entry`	ttk::entry	`tk.Widget`
`tk.Widget#ttk$Frame`	ttk::frame	`tk.Widget`
`tk.Widget#ttk$Intro`	ttk::intro	`tk.Widget`
`tk.Widget#ttk$Label`	ttk::label	`tk.Widget`
`tk.Widget#ttk$Labelframe`	ttk::labelframe	`tk.Widget`

tk.Widget#ttk$Menubutton	ttk::menubutton	tk.Widget
tk.Widget#ttk$Notebook	ttk::notebook	tk.Notebook
tk.Widget#ttk$Panedwindow	ttk::panedwindow	tk.Widget
tk.Widget#ttk$Progressbar	ttk::progressbar	tk.Widget
tk.Widget#ttk$Radiobutton	ttk::radiobutton	tk.Widget
tk.Widget#ttk$Scale	ttk::scale	tk.Widget
tk.Widget#ttk$Scrollbar	ttk::scrollbar	tk.Widget
tk.Widget#ttk$Separator	ttk::separator	tk.Widget
tk.Widget#ttk$Sizegrip	ttk::sizegrip	tk.Widget
tk.Widget#ttk$Spinbox	ttk::spinbox	tk.Widget
tk.Widget#ttk$Style	ttk::style	tk.Widget
tk.Widget#ttk$Treeview	ttk::treeview	tk.Treeview
tk.Widget#ttk$Widget	ttk::widget	tk.Widget
tk.Widget#ttk$Image	ttk::image	tk.Widget
tk.Widget#ttk$Vsapi	ttk::vsapi	tk.Widget

19.7. tk.Menu クラス

```
Menu#Separator(opts%)
Menu#Cascade(opts%) {block?}
Menu#Command(opts%) {block?}
Menu#Checkbutton(opts%) {block?}
Menu#Radiobutton(opts%) {block?}
```

19.8. tk.Canvas クラス

```
Canvas#Tag() {block?}
```

19.9. tk.CanvasItem クラス

19.19. 注意事項

Tk イメージオブジェクトは、明示的に destroy() を実行しないとメモリから削除されません。

20. データベース – sqlite3 モジュール

20.1. 関数

```
open(filename:string) {block?}
```

20.2. sqlite3 クラス

```
sqlite3.sqlite3#close()

sqlite3.sqlite3#exec(sql:string):map

sqlite3.sqlite3#getcolnames(sql:string):map {block?}

sqlite3.sqlite3#query(sql:string):map {block?}

sqlite3.sqlite3#setprop!(symbol:symbol, value):map

sqlite3.sqlite3#transaction() {block}
```

21. オーディオ

21.1. wav

22. アーカイブファイル – gzip モジュール

```
gzip.open(name:string, mode?:string, encoding:string => "utf-8")

gzip.decomp(stream:stream, winbits?:number)

gzip.comp(stream:stream, winbits?:number)


stream#gzipdecomp(winbits?:number)

stream#gzipcomp(winbits?:number)
```

23. アーカイブファイル – bzip2 モジュール

```
bzip2.open(name:string, mode?:string, encoding:string => "utf-8")

bzip2.decomp(stream:stream)

bzip2.comp(stream:stream)


stream#bzip2decomp()

stream#bzip2comp()
```

24. アーカイブファイル – zipfile モジュール

ZIP アーカイブの操作をするモジュールです。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- open 関数で ZIP アーカイブ中のファイルをオープンできるようになります。
- ストリームを受け取る引数に、ZIP アーカイブ中のファイルパス名を指定できるようになります。
- path.dir, path.walk, path.glob 関数で、ZIP アーカイブ中のディレクトリパスをサーチできるようになります。

24.1. 関数

```
zipfile.open(pathname:string) {block?}

zipfile.create(pathname:string) {block?}
```

24.2. zipfile.zipfile クラス

```
zipfile.zipfile#add(stream:stream, filename?:string):map:reduce
```

```
zipfile.zipfile#each() {block?}

zipfile.zipfile#close():reduce
```

24.3. zipfile.stat クラス

メンバ	データ型	内容
filename	string	ファイル名
comment	string	コメント
mtime	datetime	最終更新日時
crc32	number	CRC32 チェックサム
compression_method	number	圧縮形式
size	number	圧縮前のサイズ
compressed_size	number	圧縮後のサイズ
attributes	number	アトリビュート

25. アーカイブファイル – tar モジュール

TAR アーカイブの操作をするモジュールです。通常の TAR ファイルに加え、gzip で圧縮された TAR ファイル(サフィックス .tgz または.tar.gz) および bzip2 で圧縮された TAR ファイル (サフィックス.tar.bz2) も処理できます。このモジュールをインポートすることで、以下の関数の機能が拡張されます。

- open 関数で TAR アーカイブ中のファイルをオープンできるようになります。
- ストリームを受け取る引数に、TAR アーカイブ中のファイルパス名を指定できるようになります。
- path.dir, path.walk, path.glob 関数で、TAR アーカイブ中のディレクトリパスをサーチできるようになります。

25.1. 関数

```
tar.open(pathname:string) {block?}

tar.create(pathname:string) {block?}
```

25.2. tar.tar クラス

```
tar.tar#add(stream:stream, filename?:string):map:reduce

tar.tar#each() {block?}

tar.tar#close():reduce
```

25.3. tar.stat クラス

メンバ	データ型	説明

name	string	ファイル名
linkname	string	
uname	string	
gname	string	
mode	number	
uid	number	
gid	number	
size	number	
mtime	datetime	
atime	datetime	
ctime	datetime	
chksum	number	
typeflag	number	
devmajor	number	
devminor	number	

26. テキスト処理 – re モジュール

正規表現のエンジンとして鬼車（http://www.geocities.jp/kosako3/oniguruma/）を使用しています。正規表現パターンの詳細は、上記のホームページをご覧ください。

正規表現処理では、一つのファンクションで 3 つの呼び出し形式があります。状況に応じて適切な表記方法を選んでください。

関数	re.pattern メソッド	string メソッド
re.match	re.pattern#match	string#match
re.scan	re.pattern#scan	string#scan
re.split	re.pattern#split	string#splitreg
re.sub	re.pattern#sub	string#sub

26.1. 関数

```
re.match(pattern:pattern, str:string, pos:number => 0, endpos?:number):map
```

re.match インスタンスを返します。

```
re.pattern(pattern:string):map:[icase,multiline]
```

re.pattern インスタンスを返します。

```
re.scan(pattern:pattern, str:string, pos:number => 0, endpos?:number):map
    {block?}
```

```
re.split(pattern:pattern, str:string, count?:number):map {block?}

re.sub(pattern:pattern, replace, str:string, count?:number):map
```

26.2. re.match クラス

```
re.match#end(index):map

re.match#group(index):map

re.match#groups()

re.match#start(index):map
```

26.3. re.pattern クラス

```
re.pattern#match(str:string, pos:number => 0, endpos?:number):map

re.pattern#scan(str:string, pos:number => 0, endpos?:number):map {block?}

re.pattern#split(str:string, count?:number):map {block?}

re.pattern#sub(replace, str:string, count?:number):map
```

26.4. string クラスへの追加メソッド

re モジュールをインポートすると、string クラスに以下のメソッドが追加されます。

```
string#match(pattern:regex, pos:number => 0, endpos?:number):map

string#sub(pattern:regex, replace, count?:number):map

string#splitreg(pattern:regex, count?:number):map {block?}

string#scan(pattern:regex, pos:number => 0, endpos?:number):map {block?}
```

27. テキスト処理 – csv モジュール

CSV ファイルの読み書きを行います。

27.1. 関数

```
csv.parse(str:string):map

csv.read(stream:stream) {block?}

csv.write(stream:stream, elem[])

csv.writes(stream:stream, iter:iterator)
```

27.2. stream クラスへの追加メソッド

```
stream#csvread() {block?}
```

28. テキスト処理 – xml モジュール

XML ファイルの読み書きを行います。

28.1. 関数

```
xml.element(name:string, attrs%) {block?}

xml.parser() {block}

xml.read(stream:stream)
```

28.2. stream クラスへの追加メソッド

```
stream#xmlread()
```

29. テキスト処理 – yaml モジュール

YAML ファイルの読み書きを行います。YAML ファイルのフォーマットについてはオフィシャルサイト http://www.yaml.org/ を参照ください。

29.1. 関数

```
yaml.compose(obj)

yaml.parse(str:string)

yaml.read(stream:stream)

yaml.write(stream:stream, obj):reduce
```

29.2. ストリームクラスへの追加メソッド

```
stream#yamlread()
```

```
stream#yamlwrite()
```

30. プラットフォーム

30.1. win32ole

```
win32ole.connect(progid:string):map:[import_const]
win32ole.new(progid:string):map:[import_const]
```

30.2. uuid

```
uuid.generate():[upper]
```

31. 開発ツール

31.1. lets_module

コマンドラインから使用します。「AScript 開発者向けマニュアル」を参照ください。

31.2. module_builder