Schluessel ユーザリファレンス

version 0.3.2

Yuichiro Moriguchi

目次

1 Schluesselとは

Schluessel(シュリュッセル) は Java SE 5.0 以上の JRE で動作することを目的としたR5RS 準拠の Scheme のインタプリタです。R5RS の機能以外にも Java のライブラリをリフレクションを用いて使用する機能や、Scheme のプログラムを Java 側から操作する機能も備えています。

Schluessel は Scheme 言語の標準である"Revised^5 Report on the Algorithmic Language Scheme"(R5RS) に準拠しています。また、Scheme 拡張の標準である"Scheme Requests for Implementation"(SRFI) にも対応しています。

Schluessel の主な特徴として以下のことがあげられます。

数の自然な活用
: Schluessel により数を自然に活用することができます。Schluessel は以下のオブジェクトを数 (代数) の対象として扱うことができます。つまり、Schluessel を"高機能関数電卓"または"簡易数式処理システム"として使用することができます。

  - 整数/有理数
  - 半精度/単精度/倍精度浮動小数
  - 10 進浮動小数 (Decimal32/Decimal64); 仮数部は正確に 10 進数を保持します
  - 正確および不正確な複素数
  - 正確および不正確な四元数 (クオータニオン)/八元数
  - 行列/数ベクトル
  - 1 変数の多項式

豊富な Java 資産の活用
: Schluessel により Java で記述されたライブラリを活用することができます。Java のインスタンスは Scheme 変数として束縛することができます。メソッドも簡単に呼び出すことができます。また、Scheme の主要な型と Java の主要な型は自然に変換されます。

マルチバイト文字列
: Schluessel はマルチバイト文字列を自然に扱うことができます。文字については Unicode 拡張領域にも対応しています。文字列は 16 ビットの文字 (Unicode の基本多言語面) までですが (これでも実用では十分である)、Unicode 拡張領域にも対応予定です。

グラフィックスの標準サポート
: Schluessel はグラフィックスを標準でサポートします。グラフィックスは"Structure and Interpretation of Computer Programs"(SICP) にあるような図形言語により記述されます。

GUI アプリケーションの容易な記述
: Schluessel は Swing と連携した GUI アプリケーションを容易に記述できます。Java による Swing アプリケーションの記述は約束事

が多く簡単には記述できませんが、Schluessel を使用すれば 1 つのソースファイルで記述できます。

Applet との連携
Schluessel は Java Applet を連携できます。Applet に対して Schluessel は BASIC のように簡単な言語環境を用意します。

1.1 SRFI のサポート

Schluessel では以下の SRFI が使用可能です。

- SRFI-0 (Feature-based conditional expansion construct)
- SRFI-1 (リストライブラリ)
- SRFI-2 (AND-LET*)
- SRFI-4 (一様なベクタ)
- SRFI-6 (文字列ポート)
- SRFI-8 (receive)
- SRFI-9 (レコード型)
- SRFI-10 (Sharp-comma external forms)
- SRFI-13 (文字列ライブラリ)
- SRFI-14 (文字セット)
- SRFI-16 (Syntax for procedures of variable arity)
- SRFI-17 (一般化された set!)
- SRFI-18 (マルチスレッド)
- SRFI-19 (日付オブジェクト)
- SRFI-23 (エラーレポート)
- SRFI-25 (多次元配列)
- SRFI-28 (文字列のフォーマット)
- SRFI-30 (ネストされたコメント)
- SRFI-34 (例外処理)
- SRFI-38 (共有構造の表現)
- SRFI-39 (パラメータ)
- SRFI-41 (ストリーム)
- SRFI-44 (コレクション)
- SRFI-47 (配列)
- SRFI-58 (配列の記法)
- SRFI-60 (ビットとしての整数)
- SRFI-62 (S 式コメント)
- SRFI-63 (一様および一様でない配列)
- SRFI-66 (オクテットベクター)

- SRFI-67 (比較手続き)
- SRFI-69 (基本的なハッシュ表)
- SRFI-94 (型の制限された数値関数)
- SRFI-98 (環境変数へのアクセス)

2 リファレンスマニュアル

2.1 独自拡張

2.1.1 ファイルのロード

`load` ファイル名 [特殊フォーム]
戻り値: 未定義
[R5RS]Scheme ソースファイルを読み込みます。リソースファイルは変数*load-path*のリストにあるディレクトリ配下から検索され、最後にカレントディレクトリ配下のリソースが検索されます。ファイル名の拡張子".scm"は省略することができます。

`*load-path*` [変数]
load にてリソースファイルを検索するときのディレクトリのパスが格納されている変数です。変更するときは専用の手続き load-use-path を使用してください。

`add-load-path` ディレクトリ名 [*endflg*] [特殊フォーム]
戻り値: 未定義
*load-path*にパッケージパスを追加するときに使用する特殊形式です。endflg に#t を指定すると*load-path*の末尾にパスが追加されます。その他の時は*load-path*の先頭に追加されます。

`use` リソースファイルパス [特殊フォーム]
戻り値: 未定義
Java のリソースファイルとして存在する Scheme ソースファイルを読み込みます。リソースファイルは変数*use-path*のリストにあるパッケージ配下から検索され、最後にルート (デフォルト) パッケージ配下のリソースが検索されます。ファイル名の拡張子".scm"は省略することができます。

`*use-path*` [変数]
use にてリソースファイルを検索するときのパッケージのパスが格納されている変数です。変更するときは専用の特殊形式 add-use-path を使用してください。*use-path*にはデフォルトでパッケージ net.morilib.lisp.lib が登録されています。

`add-use-path` パッケージ名 [*endflg*] [特殊フォーム]
戻り値: 未定義
*use-path*にパッケージパスを追加するときに使用する特殊形式です。endflg に#t を指定すると*use-path*の末尾にパスが追加されます。その他の時は*use-path*の先頭に追加されます。

2.1.2 Java クラスの使用

. *instance method args* [特殊フォーム]
. *classname method args* [特殊フォーム]
戻り値: Java メソッドの戻り値に対応する値
Java のメソッドを呼び出します。
第 1 引数に束縛済みの変数を与えたときはその Java インスタンスのメソッドを呼び出します。
第 1 引数に未束縛のシンボルを与えたときはシンボルをクラス名として static なメソッドを呼び出します。

`new` *classname args* [特殊フォーム]
戻り値: Java インスタンス
Java クラスのコンストラクタを呼び出します。

`java-import` 内部クラス名 クラスパス [特殊フォーム]
戻り値: 未定義
クラスパスで指定された Java クラスを定義します。この形式は、

- プロパティに対しては"内部クラス名- プロパティ名"という名称のアクセサ手続き
- その他のメソッドに対しては"内部クラス名-メソッド"という名称の手続き

を自動的に生成します。

`java-new` 内部クラス名 コンストラクタ引数 ... [関数]
戻り値: Java インスタンス
Java のインスタンスを生成します。コンストラクタに与える引数は内部で Scheme のデータ型に対応する Java の型に変換されてコンストラクタに与えられます。リストは Java の配列に変換されます。

内部クラス名-プロパティ名 *Java* インスタンス [*index*] [関数]
戻り値: Java の値が Scheme 型に変換できるときは Scheme 型、それ以外は Java インスタンス
Java インスタンスのプロパティにアクセスします。たとえば、

```
package example;
public class Bean1 {
    private String prop1;
    private String[] prop2 = new String[10];
    public String getProp1() { return prop1; }
    public String setProp1(String x) { prop1 = x; }
    public String getProp2(int i) { return prop2[i]; }
    public String setProp2(int i, String x) {
        prop2[i] = x;
    }
}
```

で定義された JavaBean があり、

```
(java-import bean1 example.Bean1)
(define b (java-new bean1))
```

で Bean を定義したとするとき、prop1 の値を得るときには、

```
(bean1-prop1 b)
```

prop2 の 0 番目の値を得るときには、

```
(bean1-prop2 b 0)
```

とすれば得られます。プロパティに値をセットしたいときには一般化された set!を用います。たとえば上記の例で prop1 に文字列"abc"をセットしたいときは、

```
(set! (bean1-prop1 b) "abc")
```

prop2 の 2 番目のインデックスに文字列"string"をセットしたいときは、

```
(set! (bean1-prop1 b 2) "string")
```

とすればセットできます。

内部クラス名-メソッド名 *Java* **インスタンス 引数** ... [関数]
戻り値: Java の値が Scheme 型に変換できるときは Scheme 型、それ以外は Java インスタンス
Java インスタンスのメソッドにアクセスします。たとえば、

```
package example;
public class Class1 {
    public void print(String args...) {
        for(String s : args) {
            System.out.println(s + ":");
        }
    }
}
```

で定義された Java クラスがあり、

```
(java-import class1 example.Class1)
(define x (java-new class1))
```

でクラスを定義したとするとき、メソッド print を呼び出すには、

```
(class1-print x "a" "b" "c")
```

と記述します。引数は内部で Scheme のデータ型に対応する Java の型に変換されてコンストラクタに与えられます。リストは Java の配列に変換されます。メソッドがオーバーロードされているとき、メソッドは引数の型により選択されます。

2.1.3 format について

`format` [*output-port*] *format args . . .* [関数]
戻り値: output-port が#f または省略されたときはフォーマットされた文字列、それ以外は未定義
[SRFI-28+] フォーマットされた文字列を取得、または出力ポートに出力します。この手続きは SRFI-28 に規定されたものに加え、Common Lisp に由来するいくつかの機能を追加しています。output-port に出力ポートが指定されたときは出力ポートにフォーマットされた文字列を出力します。output-port が#f または指定されなかったときはフォーマットされた文字列を戻り値として取得します。Schluessel がサポートしているフォーマット文字列は以下の通りです。

- ~A : display によるフォーマットオプションとして最小文字数 (第 1 引数パディングされる文字長の倍数 (第 2 引数少なくとも追加されるパディング文字数 (第 3 引数パディング文字 (第 4 引数をサポートしています。
- ~S : write によるフォーマットフォーマットに write を使用すること以外~A と同じです。
- ~D, ~B, ~O, ~X : 10 進・2 進・8 進・16 進数による整数のフォーマットオプションとして最小文字数 (第 1 引数パディングされる文字長の倍数 (第 2 引数桁区切りに使用する文字 (第 3 引数 デフォルトはカンマ","区切る桁数 (第 4 引数 デフォルトは 3 をサポートしています。
- ~~ : チルダ"~"自身
- ~% : 改行文字

2.1.4 伝統的なマクロ

`define-macro` *name procedure* [特殊フォーム]
`define-macro` (*name args . . .*) *procedure* [特殊フォーム]
戻り値: 未定義
伝統的なマクロを定義します。

`macroexpand` *form* [関数]
`macroexpand-1` *form* [関数]
戻り値: マクロを展開した結果の S 式
マクロを展開した結果を得ます。macroexpand はマクロを展開した最終結果を得ます。macroexpand-1 は 1 フェーズだけマクロを展開します。

`gensym` [関数]
戻り値: 一時的に生成されたシンボル
一時的に生成されたシンボルを得ます。

2.1.5 #構文の定義

`define-sharp-symbol` *pattern expression* [関数]
戻り値: 未定義
ユーザにて#構文に関連づけられたシンボルを定義します。pattern は# 構

文を定義するパターンを指定します。パターンには以下の正規表現が使用できます。(Java のフルの正規表現は使用できません)

rs	r と s の連結
r|s	r または s
r*	r の 0 回以上の繰り返し
r+	r の 1 回以上の繰り返し
r?	r の 0 または 1 回の出現
(r)	r そのもの
.	改行以外の任意の文字
[a-zA]	文字セット
[^a-zA]	補文字セット

例えば、

```
(define-sharp-symbol "#s" 'success)
```

は、構文#s にシンボル'success を結びつけます。

`define-sharp-syntax` *pattern expression follow-exp?* [関数]
戻り値: 未定義
ユーザにて#構文に関連づけられた構文を定義します。pattern は# 構文を定義するパターンを指定します。パターンは define-sharp-symbol と同じものが指定できます。

構文はパーサにより以下のように展開されます。

```
(expression マッチした#構文の文字列 #構文に続く S 式)
```

「#構文に続く S 式」は引数 follow-exp?が#f でないときにあたえられます。(デフォルトでは#f です

例えば、

```
(define-sharp-syntax "#{.*}" (lambda (x) (display x)))
```

は、構文#{...}にマッチした構文の内容をプリントします。実行例は以下の通りです。

```
> #{aaaa}
{aaaa}#<undef>
```

```
(define-sharp-syntax "#\\?=>" (lambda (x y) (display y)) #t)
```

は、構文#?=>に続く S 式の内容をプリントします。"?"はパターンのメタ文字のためエスケープする必要があります。"\"は Scheme のメタ文字でもあるため"\"を 2 つ続ける必要があります。実行例は以下の通りです。

```
> #?=>1
1#<undef>
> #?=>(list 1 2 3)
(1 2 3)#<undef>
```

2.1.6 数学の特殊関数

`besselJ` *z nu* [関数]
戻り値: Bessel 関数 $J_\nu(z)$
次数 nu の Bessel 関数 $J_\nu(z)$ を求めます。z は複素数、ν は実数である必要があります。

`besselY` *z nu* [関数]
戻り値: Neumann 関数 $Y_\nu(z)$
次数 nu の Neumann 関数 $Y_\nu(z)$ を求めます。z は複素数、ν は実数である必要があります。

`besselI` *z nu* [関数]
戻り値: 変形 Bessel 関数 $I_\nu(z)$
次数 nu の変形 Bessel 関数 $I_\nu(z)$ を求めます。z は複素数、ν は実数である必要があります。

`besselK` *z nu* [関数]
戻り値: 変形 Bessel 関数 $K_\nu(z)$
次数 nu の変形 Bessel 関数 $K_\nu(z)$ を求めます。z は複素数、ν は実数である必要があります。

`gamma` *z* [関数]
戻り値: ガンマ関数 $\Gamma(z)$
ガンマ関数 $\Gamma(z)$ を求めます。z は複素数である必要があります。

2.1.7 その他

`#[charset]` [Reader Syntax]
*charset*で与えられた文字セットを構築します。

`nnn[.n]see` [Reader Syntax]
`nnn[.n]fee` [Reader Syntax]
`nnn[.n]dee` [Reader Syntax]
`nnn[.n]lee` [Reader Syntax]
[R5RS] 2 進浮動小数点を記述します。
指数部に

- "s"をつけると半精度
- "f"をつけると単精度
- "d","l"をつけると倍精度 (4 倍精度はサポートしていません)

となります。

`nnn[.n]dfee` [Reader Syntax]
`nnn[.n]ddee` [Reader Syntax]
10 進浮動小数点を記述します。
指数部に

- "df"をつけると Decimal32
- "dd"をつけると Decimal64

となります。
Decimal32 は $\pm 0.000000 \times 10^{-95}$ から $\pm 9.999999 \times 10^{96}$ まで、
Decimal64 は $\pm 0.000000000000000 \times 10^{-383}$ から $\pm 9.999999999999999 \times 10^{384}$ までの 10 進小数を扱えます。

2.2 正規表現

正規表現に関するライブラリです。ここに記述されている手続きでは、Java 標準の正規表現が使用できます。

`#/regexp/[uic]` [Reader syntax]
戻り値: 正規表現オブジェクト
正規表現オブジェクトを構築します。末尾に以下のフラグを追加できます。フラグは複数指定可能です。

u	Unicode を使用します
i	大文字/小文字を区別しません
c	標準等価を有効にします

`regexp?` *datum* [関数]
戻り値: 正規表現オブジェクトのとき#t
与えられたオブジェクトが正規表現オブジェクトであるかを調べます。

`regexp-replace` *regexp dest-string replace-string* [関数]
戻り値: 置き換えられた文字列
与えられた正規表現に従い、文字列を置き換えます。最初に一致した文字のみ置き換えられます。正規表現は文字列または正規表現オブジェクトで与えます。

`regexp-replace-all` *regexp dest-string replace-string* [関数]
戻り値: 置き換えられた文字列
与えられた正規表現に従い、文字列を置き換えます。一致した文字すべてが置き換えられます。正規表現は文字列または正規表現オブジェクトで与えます。

`regexp->string` *regexp-object* [関数]
戻り値: 正規表現を表す文字列
正規表現オブジェクトの文字列表現を取得します。

`string->regexp` *regexp* [関数]
戻り値: 正規表現オブジェクト
文字列を正規表現オブジェクトに変換します。

`rxmatch` *regexp-object dest-string* [関数]
戻り値: マッチした文字列; グループ化されているときはグループにマッチした文字列が多値として返す
文字列をマッチし、その結果を多値として返します。

`rxmatch-list` *regexp-object dest-string* [関数]
戻り値: マッチした文字列; グループ化されているときはグループにマッチした文字列をとして返す
文字列をマッチし、その結果をリストとして返します。

`matchrx?` *regexp-object dest-string* [関数]
戻り値: マッチしたときに#t
文字列が正規表現にマッチするかを調べます。

2.3 線形代数ライブラリ

行列を操作する手続きです。

`list->matrix` ((*x* ...) ...) [関数]
戻り値: 行列
リストから行列を生成します。
次の手続き呼び出しは

```
(list->matrix '((1 2 3) (4 5 6)))
```

行列

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

を返します。

`matrix?` *obj* [関数]
戻り値: obj が行列のとき#t
オブジェクトが行列型であるかを判定します。

`list->number-vector` (*x* ...) [関数]
戻り値: 数値ベクトル
リストから数値ベクトルを生成します。
次の手続き呼び出しは

```
(list->number-vector '(1 2 3))
```

ベクトル (1 2 3) を返します。

`number-vector` *x* ... [関数]
戻り値: 数値ベクトル
引数から数値ベクトルを生成します。
次の手続き呼び出しは

```
(number-vector 1 2 3)
```

ベクトル (1 2 3) を返します。

`make-matrix` *rowsize colsize* [*fill*] [関数]
戻り値: 行列
一様な値を持つ行列を生成します。*fill*のデフォルト値は 0 とします。
次の手続き呼び出しは

```
(make-matrix 3 4 1)
```

行列

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

を返します。

`make-number-vector` *size* [*fill*] [関数]
戻り値: 数値ベクトル
一様な値を持つ数値ベクトルを生成します。
次の手続き呼び出しは

```
(make-number-vector 3 1)
```

ベクトル (1 1 1) を返します。

`matrix-determinant` *matrix* [関数]
戻り値: 数値
行列の行列式を求めます。

`matrix=` *matrix1 matrix2* [関数]
戻り値: 行列が等しいとき#t
行列が等しいかを調べます。数の比較には=を使用します。

`matrix+` *matrix ...* [関数]
戻り値: 行列
行列の和を求めます。

`matrix-` *matrix ...* [関数]
戻り値: 行列
行列の差を求めます。

`matrix*` *matrix ...* [関数]
戻り値: 行列
行列の積を求めます。

`matrix-invert` *matrix ...* [関数]
戻り値: 行列
逆行列を求めます。

`lu-decompose` *matrix* [関数]
戻り値: 置換行列、L 行列、U 行列の 3 値
行列を LU 分解し、$P(LU)$ の形にします。

`solve-linear-equation` *matrix vector* [関数]
戻り値: 連立 1 次方程式の解
方程式 $A\vec{x} = \vec{b}$ を解きます。

`matrix-row-size` *matrix* [関数]
`matrix-column-size` *matrix* [関数]
戻り値: 整数
行列 *matrix*の行 (列) のサイズを返します。

`matrix-row-vector` *matrix n* [関数]
`matrix-column-vector` *matrix n* [関数]
戻り値: ベクトル
行列 *matrix*の *n*番目の行 (列) ベクトルを返します。

`matrix-transpose` *matrix* [関数]
戻り値: 転置行列 A^t
行列 *matrix*の転置行列を返します。

2.4 四元数・八元数

四元数･八元数についてです。四元数 $a+bi+cj+dk$ を入力するには、a+bi+cj+dk とします。

八元数 $a + bi + cj + dk + el + fil + gjl + hkl$ を入力するには a+bi+cj+dk+eo+fio+gjo+hko とします。(虚数単位 l の代わりに o を使用します。)

`+` *n ...* [関数]
戻り値: 数値
[R5RS+] 数値の和を求めます。この手続きは四元数・八元数に拡張されています。

`-` *n ...* [関数]
戻り値: 数値
[R5RS+] 数値の差を求めます。この手続きは四元数・八元数に拡張されています。

`*` *n ...* [関数]
戻り値: 数値
[R5RS+] 数値の積を求めます。この手続きは四元数・八元数に拡張されています。

`/` *n ...* [関数]
戻り値: 数値
[R5RS+] 数値の商を求めます。この手続きは四元数・八元数に拡張されています。

`quaternion?` *obj* [関数]
戻り値: オブジェクトが四元数のときに#t

`octonion?` *obj* [関数]
戻り値: オブジェクトが八元数のときに#t

2.5 1 変数の多項式

1 変数の多項式を計算します。

`polynomial1` *n ...* [関数]
戻り値: 1 変数多項式
1 変数の多項式を生成します。

```
(polynomial1 1 2 3 4)
```

は多項式 $x^3 + 2x^2 + 3x + 4$ を表します。

`differenciate1` *poly* [関数]
戻り値: 1 変数多項式
多項式を微分します。

`integrate1` *poly begin end* [関数]
戻り値: 数値
多項式の区間 [*begin end*] での定積分を求めます。

`substitute1` *poly x* [関数]
戻り値: 数値
多項式に x を代入した値を求めます。

`+` *n ...* [関数]
戻り値: 1 変数多項式
[R5RS+] 多項式の和を求めます。この手続きは多項式に拡張されています。

`-` *n ...* [関数]
戻り値: 1 変数多項式
[R5RS+] 多項式の差を求めます。この手続きは多項式に拡張されています。

`*` *n ...* [関数]
戻り値: 1 変数多項式
[R5RS+] 多項式の積を求めます。この手続きは多項式に拡張されています。

`quotient` *n m* [関数]
戻り値: 1 変数多項式
[R5RS+] 多項式の商を求めます。この手続きは多項式に拡張されています。

`remainder` *n m* [関数]
戻り値: 数値
[R5RS+] 多項式の剰余を求めます。この手続きは多項式に拡張されています。

`polynomial1?` *obj* [関数]
戻り値: オブジェクトが多項式のときに#t

`degree-polynomial1` *poly* [関数]
戻り値: 整数
多項式の次数を求めます。0 の次数は-1 です。

`integrate-polynomial1` *poly C* [関数]
戻り値: 1 変数多項式
多項式を不定積分します。積分定数は *C*で与えます。

`polynomial1-ref` *poly n* [関数]
戻り値: 数値
多項式の *n*次の項の数値を返します。次数が多項式の次数を超えたときには 0 を返します。

2.6 実数の区間

実数の区間を扱うライブラリです。

`#R[n` *m*] [Reader Syntax]
`#R(n` *m*] [Reader Syntax]
`#R[n` *m*) [Reader Syntax]
`#R(n` *m*) [Reader Syntax]
それぞれ実数の

- 閉区間 $[n, m]$
- 半開区間 $(n, m]$
- 半開区間 $[n, m)$
- 開区間 (n, m)

を記述します。

`make-closed-interval` *n m* [関数]
`make-left-open-interval` *n m* [関数]
`make-right-open-interval` *n m* [関数]
`make-open-interval` *n m* [関数]
それぞれ実数の

- 閉区間 $[n, m]$
- 半開区間 $(n, m]$
- 半開区間 $[n, m)$
- 開区間 (n, m)

を生成します。

`neighborhood-of?` *range obj* [関数]
戻り値: オブジェクトが区間にあるときは#t
オブジェクトが区間 *range*にあるかを調べます。

`topology-closed?` *range* [関数]
`topology-open?` *range* [関数]
戻り値: 区間が閉じている (開いている) とき#t
区間 *range*が閉じている (開いている) かを調べます。

`topology-contained?` *range1 range2* [関数]
戻り値: $range1 \subseteq range2$ のとき#t
*range1*が *range2*に含まれるとき#t を返します。

`topology-independent?` *range1 range2* [関数]
戻り値: $range1 \cap range2 = \phi$ のとき#t
*range1*と *range2*が独立しているとき#t を返します。

`topology-empty?` *range* [関数]
戻り値: $range = \phi$ のとき#t
*range*が空集合であるかを調べます。

`set-maximum` *range* [関数]
`set-minimum` *range* [関数]
戻り値: $\max range(\min range)$
*range*の最大値 (最小値) を求めます。最大値 (最小値) のないときは#f を返します。
最大値 (最小値) のない場合として、

- 最大値 (最小値) が無限大
- 最大 (最小) の端が開いている

があります。

`set-supremum` *range* [関数]
`set-infimum` *range* [関数]
戻り値: $\sup range(\inf range)$
*range*の上界 (下界) を求めます。上界 (下界) のないときは#f を返します。
上界 (下界) のない場合として、上界 (下界) が無限大の場合があります。
(最大 (最小) の端が開いているときも上界 (下界) はあります。)

`topology-union` *range* ... [関数]
戻り値: $range_1 \cup range_2 \cup \cdots$
range...の和集合を求めます。

`topology-intersection` *range* ... [関数]
戻り値: $range_1 \cap range_2 \cap \cdots$
range...の共通部分を求めます。

`topology-closure` R [関数]
`topology-interior` R [関数]
戻り値: $\overline{R}(R^\circ)$
Rの (集合論的な) 閉包 (内部) を求めます。

`topology-cardinality` *range* [関数]
戻り値: *range*の濃度
*range*の濃度 (cardinality) を求めます。

`cardinality->integer` *cardinality* [関数]
戻り値: 整数
*cardinality*を整数に変換します。無限の濃度のときは#f を返します。

`cardinality-finite?` *cardinality* [関数]
戻り値: 濃度が有限であるときに#t
*cardinality*が有限のときに#t を返します。

`cardinality-countable?` *cardinality* [関数]
戻り値: 濃度が可算であるときに#t
*cardinality*が可算、つまり $\aleph_0$ 以下のときに#t を返します。

`+` *cardinality* ... [関数]
戻り値: 濃度
[R5RS+] 濃度の和を求めます。この手続きは濃度に拡張されています。

`*` *cardinality* ... [関数]
戻り値: 濃度
[R5RS+] 濃度の積を求めます。この手続きは濃度に拡張されています。

2.7 非数値の行列ライブラリ

非数値要素の行列を操作する手続きです。数値行列は非数値行列のサブセットと見なされます。

`list->datum-matrix` ((*x* ...) ...) [関数]
戻り値: 行列
リストから非数値行列を生成します。

`datum-matrix?` *obj* [関数]
戻り値: obj が行列のとき#t
オブジェクトが行列型であるかを判定します。

`make-datun-matrix` *rowsize colsize* [*fill*] [関数]
戻り値: 行列
一様な値を持つ行列を生成します。*fill*のデフォルト値は未定義値とします。

`matrix-row-size` *matrix* [関数]
`matrix-column-size` *matrix* [関数]
戻り値: 整数
行列 *matrix*の行 (列) のサイズを返します。

`matrix-row-vector` *matrix n* [関数]
`matrix-column-vector` *matrix n* [関数]
戻り値: ベクトル
行列 *matrix*の *n*番目の行 (列) ベクトルを返します。

`matrix-transpose` *matrix* [関数]
戻り値: 転置行列 A^t
行列 *matrix*の転置行列を返します。

`cons-tensor-product` *vec1 vec2* [関数]
戻り値: 行列
Scheme ベクトル vec1 と vec2 の「テンソル積」を返します。ここで、テンソル積は次のように定義します
A_{ij} =(cons $vec1_i$ $vec2_j$)

2.8 ファイル操作

ファイルを操作する手続きです。

`chdir` [*dir*] [関数]
戻り値: 引数がないときは現在の作業ディレクトリ、引数があるときは未定義
現在の作業ディレクトリを変更します。

`directory?` *path* [関数]
戻り値: ディレクトリならば#t
与えられたパスが存在するディレクトリであるかを調べます。

`file?` *path* [関数]
戻り値: ファイルならば#t
与えられたパスが存在するファイルであるかを調べます。

`exist?` *path* [関数]
戻り値: 存在すれば#t
与えられたパスが存在するかを調べます。

`directory-list` *dir* [*wildcard-filter-expr*] [関数]
戻り値: ディレクトリに存在するファイルのリスト
ディレクトリに存在するファイルを取得します。この手続きはディレクトリを再帰的に評価しません。
wildcard-filter-expr には以下の式が指定できます。

(and [expr . . .])	与えられた条件をすべてみたすとき選択されます
(or [expr . . .])	与えられた条件のいずれかをみたすとき選択されます
(not expr)	与えられた条件を満たさないとき選択されます
exist	存在するパスを選択します
directory	存在するディレクトリを選択します

file	存在するファイルを選択します
文字列	文字列で与えられたワイルドカードに一致するパスを選択します。 ワイルドカードのメタ文字は次の通りです。 * 任意の 0 文字以上 . 任意の 1 文字
[...]	文字集合

`directory-fold` *path proc seed* [*wild-card*] [関数]
戻り値: ファイルならば#t
与えられたパスのサブディレクトリにある (ワイルドカードでフィルタされた) ファイル名すべてに対して fold します。proc は 2 つの引数を持ち、第 1 引数はファイル名、第 2 引数は今までの結果が与えられます。

`make-path-name` *path file* [関数]
戻り値: 新しいパス
パス名とファイル名を組み合わせて新しいパスを生成します。この手続きを使用すれば OS のパス区切りの依存性を回避することができます。

`expand-path` *path* [関数]
戻り値: 文字列
パスを絶対パスに変換します。

`absolute-path?` *path* [関数]
戻り値: パスが絶対パスのとき#t
パスが絶対パスかを調べます。

`relative-path?` *path* [関数]
戻り値: パスが相対パスのとき#t
パスが相対パスかを調べます。

`file-mtime=?` *time1 time2* [関数]
戻り値: 下記参照
ファイルの更新日時が等しいかを調べます。引数にはファイルパス･SRFI-19 形式の時間オブジェクト・UNIX エポックからの秒数指定できます。

`file-mtime>=?` *time1 time2* [関数]
戻り値: 下記参照
ファイルの更新日時が同じまたは未来であるかを調べます。

`file-mtime>?` *time1 time2* [関数]
戻り値: 下記参照
ファイルの更新日時が未来であるかを調べます。

`file-mtime<=?` *time1 time2* [関数]
戻り値: 下記参照
ファイルの更新日時が同じまたは過去であるかを調べます。

`file-mtime<?` *time1 time2* [関数]
戻り値: 下記参照
ファイルの更新日時が過去であるかを調べます。

`mkdir` *path* [関数]
戻り値: ディレクトリが作成されれば#t
ディレクトリを作成します。

`copy-file` *src-file dest-file* [関数]
戻り値: 未定義
ファイルをコピーします。

`move-file` *src-file dest-file* [関数]
戻り値: 移動が正常終了すれば#t ファイルを移動します。

`remove-file` *file ...* [関数]
戻り値: 未定義
ファイルを削除します。

`rmdir` *path* [関数]
戻り値: ディレクトリを削除できれば#t
ディレクトリを削除します。

`touch-file` *file ...* [関数]
戻り値: 未定義
ファイルを touch します。

2.9 XML 機能

XML 形式のファイルを SXML 形式の S 式に変換する機能です。現時点では XML → SXML の変換のみサポートしています。

SXML とは、XML を S 式で表現するための仕様です。SXML の仕様は以下のサイトにあります。
http://okmij.org/ftp/Scheme/xml.html

`xml->sxml` *input-port* [関数]
戻り値: SXML
XML 形式のファイルを SXML に変換します。XML のパースには Java 標準の SAX パーサを使用しています。

2.10 形式文法によるパーサ生成

形式文法を使用して、ファイルや文字列を解析するパーサを生成することができます。
このライブラリでは、LALR(1) 文法のクラスにある形式文法のパーサを生成

することができます。
正規表現を使用して文字列をトークナイズする字句解析器も生成できます。
このパーサで使用する字句解析器は終端記号を表すシンボルと semantics な解析結果の 2 値を返す手続きならばどのようなものでも使用できます。
使用できる正規表現は以下の通りです。

rs	r と s の連結
r|s	r または s
r*	r の 0 回以上の繰り返し
r+	r の 1 回以上の繰り返し
r?	r の 0 または 1 回の出現
(r)	r そのもの
.	改行以外の任意の文字
[a-zA]	文字セット
[^a-zA]	補文字セット

(注) Java 標準の正規表現は使用できません。

`make-grammar` *grammar* [関数]
戻り値: 解析された形式文法オブジェクト
与えられた形式文法を解析します。解析された文法は parse-grammar 手続きで使用することができます。
形式文法は、以下のような形式で与えます。

```
((生成規則の左辺（生成規則の右辺 ...) reduce されたときに実行される手続き
 ...)
```

終端記号および非終端記号はシンボルで与えます。開始記号はシンボル S(大文字) で与えられます。
"reduce されたときに実行される手続き"は右辺の semantics な解析結果を引数として呼び出されます。
例えば、

```
S → S + T
S → S - T
S → T
T → T * F
T → T / F
T → F
F → lparen S rparen
F → num
```

という文法 (この文法は一般的な四則演算を定義します) を定義し、この文法に従って計算するときの文法の定義は、

```
(make-grammar
  `((S (S + T)     ,(lambda (t1 x t2) (+ t1 t2)))
    (S (S - T)     ,(lambda (t1 x t2) (- t1 t2)))
    (S (T)         ,(lambda (t1) t1))
```

```
(T (T * F)     ,(lambda (f1 x f2) (* f1 f2)))
(T (T / F)     ,(lambda (f1 x f2) (* f1 f2)))
(T (F)         ,(lambda (f1) f1))
(F (lparen S rparen)
   ,(lambda (x s1 y) s1))
(F (num)
   ,(lambda (v1) (string->number v1)))))
```

のように指定します。シンボル lparen, rparen, num は字句解析器により与えられます。
文法に"競合"(conflict) が発生してもこの手続きはエラー・警告を出しません。競合を取得したいときは get-conflicts 手続きを使用してください。

`parse-grammar` *parser-object lexer* [関数]
戻り値: パース結果の semantics のスタックトップ
与えられた字句解析器を使用してパースします。
字句解析器は終端記号を表すシンボルと semantics な解析結果の 2 値を返す手続きです。

`get-conflicts` *parser-object* [関数]
戻り値: 競合のリスト
文法における"競合"を取得します。競合は以下の形で取得されます。

```
(...
 (shift-reduce  シフトする終端記号
                還元する文法規則)   ; シフト-還元競合
 (reduce-reduce 還元する文法規則
                還元する文法規則)   ; 還元-還元競合
 ...)
```

文法規則は (生成規則の左辺 (生成規則の右辺 ...)) の形のリストで取得されます。

`make-regexp-tokenize-pattern` *lexer-patterns* [関数]
戻り値: トークナイザパターン
正規表現によるトークナイザパターンを生成します。パターンは以下の形で与えます。

```
((正規表現を表す文字列 . 終端記号を表すシンボル)
 ...)
```

例:四則演算子と数値を解析するトークナイザ

```
(make-regexp-tokenize-pattern
    '(("[0-9]+(\\.[0-9]+)?" . num
      ("[
      ]+" . whitespace)  ; 空白+タブ+改行
      ("\\(" . lparen
      ("\\)" . rparen
```

```
("\\+" . +
("-" . -
("\\*" . *
("\\/" . /))
```

`make-regexp-tokenizer-string` *tokenizer-pattern string* [関数]
戻り値: トークナイザ
文字列に対してパターンを適用するトークナイザを取得します。

`make-regexp-tokenizer-port` *tokenizer-pattern iport* [関数]
戻り値: トークナイザ
入力ポートに対してパターンを適用するトークナイザを取得します。

`tokenize` *tokenizer* [関数]
戻り値: 終端記号のシンボルと一致した部分の文字列の 2 値
トークナイザを用いて入力を区切り、その結果を返します。

2.11 図形言語・グラフィックス機能

`make-window-frame` *xsize ysize* [*x-begin y-begin x-end y-end*] [関数]
戻り値: ウィンドウフレームオブジェクト
サイズ xsize × ysize のウィンドウを生成します。3 番目から 6 番目の引数により、ウィンドウに付随する座標系を指定することができます。3〜6 番目の引数を指定したときの座標系は数学で使用する座標系 (x 軸は右に行くほど大きく、y 軸は上に行くほど大きい) となります。3〜6 番目の引数を指定しなかったときはコンピュータグラフィックで使用される座標系 (x 軸は右に行くほど大きく、y 軸は下に行くほど大きい) となります。3 番目と 4 番目の引数には左下の座標、5 番目と 6 番目の引数には右上の座標を指定します。

ウィンドウをリサイズしたときにも付随する座標系は保存され、新しい座標のマッピングで再描画されます。ウィンドウがクローズされたときにはウィンドウフレームは破棄されます。クローズしたウィンドウに描画をしようとしたときには何も起きません。

`make-painter-frame` *xsize ysize* [*x-begin y-begin x-end y-end*] [関数]
戻り値: ペインタフレームオブジェクト
サイズ xsize × ysize のペインタフレームを生成します。引数については make-window-frame と同じです。ペインタフレームについてはサイズを変更しても変わりません。

`load-image-painer` *image-file-name* [関数]
戻り値: 画像ペインタオブジェクト
画像ファイルを画像ペインタとして読み込みます。読み込める画像ファイルの形式は JPEG, GIF, PNG です。

`below-painter` *above below* [関数]
戻り値: 合成されたペインタオブジェクト
画像ペインタを上下に並べます。画像サイズは y 軸方向に半分縮小されます。

`beside-painter` *left right* [関数]
戻り値: 合成されたペインタオブジェクト
画像ペインタを左右に並べます。画像サイズは x 軸方向に半分縮小されます。

`display-painter` *frame painter x y* [関数]
戻り値: 未定義
ペインタをフレームに描画します。座標はフレームに付随する座標で指定します。

`draw-line` *frame begin-x begin-y end-x end-y* [関数]
戻り値: 未定義
フレームに線を描画します。座標はフレームに指定された座標系が適用されます。

`draw-string` *frame string x y* [関数]
戻り値: 未定義
フレームに文字列を描画します。座標はフレームに指定された座標系が適用されます。

`draw-rectangle` *frame left-top-x left-top-y right-bottom-x right-bottom-y* [関数]
戻り値: 未定義
フレームに矩形を描画します。矩形の左上の頂点と右下の頂点の座標を指定します。座標はフレームに指定された座標系が適用されます。

`draw-oval` *frame left-top-x left-top-y right-bottom-x right-bottom-y* [関数]
戻り値: 未定義
フレームに矩形に接する楕円 (円) を描画します。矩形の左上の頂点と右下の頂点の座標を指定します。座標はフレームに指定された座標系が適用されます。

`fill-rectangle` *frame left-top-x left-top-y right-bottom-x right-bottom-y* [関数]
戻り値: 未定義
フレームに矩形を描画し、その内部を塗りつぶします。矩形の左上の頂点と右下の頂点の座標を指定します。座標はフレームに指定された座標系が適用されます。

`fill-oval` *frame left-top-x left-top-y right-bottom-x right-bottom-y* [関数]
戻り値: 未定義
フレームに矩形に接する楕円 (円) を描画し、その内部を塗りつぶします。矩形の左上の頂点と右下の頂点の座標を指定します。座標はフレームに指定された座標系が適用されます。

`clear-rectangle` *frame left-top-x left-top-y right-bottom-x right-bottom-y* [関数]
戻り値: 未定義
フレームから矩形で指定された領域をクリア-バックグラウンドカラーで塗りつぶします。矩形の左上の頂点と右下の頂点の座標を指定します。座標はフレームに指定された座標系が適用されます。

`clear-screen` *frame* [関数]
戻り値: 未定義
フレームをクリアします。ウィンドウフレームのときは描画したオブジェクト情報をクリアします。ペインタフレームのときはバックグラウンドカラーで塗りつぶします。

`set-current-color!` *frame color* [関数]
戻り値: 未定義
フレームのフォアグラウンドカラーを指定した色オブジェクトに指定します。

`make-color` *red green blue* [*alpha*] [関数]
戻り値: 色オブジェクト
色オブジェクトを生成します。引数には R,G,B, アルファ値を 0.0〜1.0 の範囲で指定します。

`*color-black*` [変数]
`*color-blue*` [変数]
`*color-cyan*` [変数]
`*color-dark-gray*` [変数]
`*color-gray*` [変数]
`*color-green*` [変数]
`*color-light-gray*` [変数]
`*color-magenta*` [変数]
`*color-orange*` [変数]
`*color-pink*` [変数]
`*color-red*` [変数]
`*color-white*` [変数]
`*color-yellow*` [変数]
定義済みの色オブジェクトです。

2.12 Swing 連携機能

Swing と連携して Scheme で GUI を構築するためのライブラリです。

`make-button` *text action* [関数]
戻り値: ボタンコンポーネント
指定された text のボタンを生成します。ボタンが押下されたときに実行される手続きを action で指定します。この手続きは引数を 1 つもつ必要があります。

`make-button-with-action` *accelerator-key: accelerator-key long-description: long-description mnemonic-key: mnemonic-key name: name short-description: short-description small-icon: small-icon* [関数]
戻り値: ボタンコンポーネント
アクションを伴うボタンを生成します。
アクションの意味は以下の通りです。キーを指定するシンボルについてはこちらを参照してください。

action	アクションを指定する手続きを指定します。アクション手続きの引数は 1 つです
accelerator-key	アクセラレータキーを指定するシンボルを指定します
long-description	長い記述を指定します
mnemonic-key	ニーモニックキーを指定します
name	名称を指定します
short-description	短い記述を指定します
small-icon	アイコンを指定します

`make-canvas` [*x1 y1 x2 y2*] [関数]
戻り値: キャンバスコンポーネント
ペインタフレームワークにより描画できるキャンバスを生成します。引数がないときはシステムの座標系に従います。引数が与えられているときはキャンバスの左上を (x1, y1)、右下を (x2, y2) とした座標系に変換します。そのとき、座標系の方向は数学等で使用されているものになります。(通常コンピュータで使用されている座標系とは異なります

`make-label` *name* [関数]
戻り値: ラベルコンポーネント
name のテキストをもつラベルを生成します。

`make-panel` [関数]
戻り値: パネルコンポーネント
パネルを生成します。パネルは Swing オブジェクトを複数格納するコンテナとして使用できます。

`make-text-field` *size* [*text*] [関数]
戻り値: テキストフィールドコンポーネント
指定されたテキストサイズをもつテキストフィールドを生成します。 text で指定された初期テキストを指定することもできます。

`make-text-area` [*row col*] [関数]
戻り値: テキストエリアコンポーネント
指定された行幅と列幅をもつテキストエリアを生成します。

`make-window` *title* [*xsize ysize* [*menus*]] [関数]
戻り値: ウィンドウ (フレーム) コンポーネント
ウィンドウ (フレーム) を生成します。この呼び出しでウィンドウは表示されません。menus にてメニューバーを指定できます。指定方法は以下の通りです。

("menu1" proc)	1 つの引数をもつ proc を実行するメニューです。
("menu1" proc key key2)	1 つの引数をもつ proc を実行するメニューです。ショートカットキーが使用できます
("menu1" seperator)	メニュー区切りです
("menu1" (submenu))	子メニューを指定します

`exec-window` *window* [関数]
戻り値: 未定義
ウィンドウを表示します。ウィンドウが閉じられたときにはプログラムを終了します。

`make-modal-dialog` *parent-window title x y* [関数]
戻り値: ダイアログコンポーネント
モーダルダイアログコンポーネントを生成します。この呼び出しでダイアログは表示されません。show-window を使用してください。

`make-modeless-dialog` *parent-window title x y* [関数]
戻り値: ダイアログコンポーネント
モードレスダイアログコンポーネントを生成します。この呼び出しでダイアログは表示されません。show-window を使用してください。

`get-text` *has-text* [関数]
戻り値: 文字列
テキストフィールド等のテキストを持つコンポーネントからテキストを取り出します。

`set-text!` *has-text str* [関数]
戻り値: 未定義
テキストフィールド等のテキストを持つコンポーネントにテキストを設定します。

`make-combobox` *item-list* [関数]
戻り値: コンボボックスコンポーネント
指定された item-list のリストの内容を持つコンボボックスを生成します。値は編集可能です。リストは (表示される値 . get-selected-item-value で取得される値) の alist です。

`make-listbox` *item-list* [関数]
戻り値: リストボックスコンポーネント
指定された item-list のリストの内容を持つコンボボックスを生成します。値は編集不可能です。リストは (表示される値 . オブジェクトとして取得される値) の alist です。

`get-selected-item-value` *selectable* [関数]
戻り値: Scheme オブジェクト
リストボックスやコンボボックス等の選択可能なコンポーネントから値を取得します。

`show-yes-no-dialog` *parent-component message title message-type* [関数]
戻り値: はいのときは#t、いいえのときは#f
はい/いいえを選択するダイアログを表示します。parent-component は親コンポーネントを設定します。親コンポーネントを指定しないときは#f を指定します。message-type はエラー・警告・情報等のメッセージ種類をシンボルで指定します。
指定できる種類は以下の通りです。

- error: エラー
- information: 情報
- warning: 警告
- question: 質問
- plain: アイコンなし

`show-yes-no-cancel-dialog` *parent-component message title message-type* [関数]
戻り値: はいのときは#t、いいえのときは#f、キャンセルされたときはシンボル canceled
はい/いいえ/キャンセルを選択するダイアログを表示します。

`show-input-dialog` *parent-component message title message-type* [関数]
戻り値: 入力された文字列、キャンセルされたときは#f
文字列の入力を促すダイアログを表示します。

`show-message-dialog` *parent-component message title message-type* [関数]
戻り値: 未定義
メッセージを表示するダイアログを表示します。

`make-checkbox` *text* [関数]
戻り値: チェックボックスコンポーネント
チェックボックスコンポーネントを生成します。

`checkbox-checked?` *checkbox* [関数]
戻り値: チェックされていれば#t、そうでなければ#f
チェックボックスにチェック状態を取得します。

`set-checkbox-checked!` *checkbox check* [関数]
戻り値: 未定義
チェックボックスのチェック状態を変更します。

`make-file-chooser` *extension-filters* [関数]
戻り値: ファイルチューザコンポーネント
ファイルチューザを生成します。引数 extension-filters でフィルタリングする拡張子のリストを与えます。書式は以下の通りです。

```
(("拡張子 [,拡張子...]" . ファイルチューザに表示する文字列
 ...)
```

この手続きの呼び出しではファイルチューザは表示されません。show-xxx-file-chooser を呼び出してください。

`set-file-selection-mode!` *filechooser mode* [関数]
戻り値: 未定義
ファイルチューザの取得モードを変更します。モードは以下の通りです。

files	ファイルのみ
directories	ディレクトリのみ
files-and-directories	ファイルとディレクトリ

`set-accept-all-file-filter-used!` *filechooser use?* [関数]
戻り値: 未定義
「すべてのファイルを選択する」をフィルタとして使用するかを設定します。

`show-open-file-chooser` *filechooser parent-component* [関数]
戻り値: 選択されたファイル、キャンセルされたときは#f
ファイルチューザを「開く」として表示します。

`show-save-file-chooser` *filechooser parent-component* [関数]
戻り値: 選択されたファイル、キャンセルされたときは#f
ファイルチューザを「保存」として表示します。

`show-file-chooser` *filechooser parent-component* [関数]
approve-button-text
戻り値: 選択されたファイル、キャンセルされたときは#f
ファイルチューザを表示します。選択時のボタンを approve-button-text で指定します。

`show-window` *window* [関数]
戻り値: 未定義
ウィンドウを表示します。

`hide-window` *window* [関数]
戻り値: 未定義
ウィンドウを隠します。

`set-dialog-resizable!` *dialog resize?* [関数]
戻り値: 未定義
ダイアログのサイズ変更を許可するかを指定します。

`make-menu-item` *title action-proc* [*mnenomic-key mnenomic-key-modifier*] [関数]
戻り値: メニューアイテムコンポーネント
メニューアイテムコンポーネントを生成します。action-proc は 1 つの引数を持つ手続きを与えます。mnenomic-key mnenomic-key-modifier はニーモニックキーを与えます。キーを指定するシンボルについてはこちらを参照してください。

`make-radio-button` *text* [*action-proc*] [関数]
戻り値: ラジオボタンコンポーネント
ラジオボタンコンポーネントを生成します。action-proc は 1 つの引数を持つ手続きを与えます。

`make-button-group` *radio-button-list* [関数]
戻り値: ボタングループ
ボタングループを生成します。グループに含まれるボタンは radio-button-list で指定します。

`copy-to-system-clipboard!` *datum* [関数]
戻り値: 未定義
指定された Scheme オブジェクトをシステムクリップボードに転送します。

`get-gui-element-width` *gui-component* [関数]
戻り値: GUI コンポーネントの幅
指定された GUI コンポーネントの幅を取得します。

`get-gui-element-height` *gui-component* [関数]
戻り値: GUI コンポーネントの高さ
指定された GUI コンポーネントの幅を取得します。

`set-background-color!` *gui-component color* [関数]
戻り値: 未定義
指定された GUI コンポーネントの背景色を設定します。

`set-preferred-width!` *gui-component width* [関数]
戻り値: 未定義
指定された GUI コンポーネントの幅の推奨値を設定します。

`set-preferred-height!` *gui-component height* [関数]
戻り値: 未定義
指定された GUI コンポーネントの高さの推奨値を設定します。

`get-bounds-x` *gui-component* [関数]
戻り値: コンポーネントの X 座標
指定された GUI コンポーネントの X 座標を取得します。

`get-bounds-y` *gui-component* [関数]
戻り値: コンポーネントの Y 座標
指定された GUI コンポーネントの Y 座標を取得します。

`set-bounds-x!` *gui-component x* [関数]
戻り値: 未定義
指定された GUI コンポーネントの X 座標を設定します。

`set-bounds-y!` *gui-component y* [関数]
戻り値: 未定義
指定された GUI コンポーネントの Y 座標を設定します。

`get-selected-button-value` *button-group* [関数]
戻り値: ボタングループで選択された値
指定されたボタングループで選択された値を取得します。

`set-selected-button-value!` *button-group datum* [関数]
戻り値: 未定義
指定されたボタングループの選択されている値を変更します。

`make-horizontal-scroll-bar` *value extent min max* [関数]
[*real-min real-max*]
戻り値: 水平スクロールバーコンポーネント
水平スクロールバーコンポーネントを生成します。引数は以下のような意味を持ちます。real-min, real-max が指定されているときはその値でスケールされます。

value	初期値
extent	増分
min	最小値
max	最大値
real-min	スケールされた最小値
real-max	スケールされた最小値

`make-vertical-scroll-bar` *value extent min max* [*real-min* [関数]
real-max]
戻り値: 垂直スクロールバーコンポーネント
水平スクロールバーコンポーネントを生成します。

`get-scroll-bar-value` *scroll-bar* [関数]
戻り値: スクロールバーの値
スクロールバーの示す値を取得します。

`get-scroll-bar-relative-value` *scroll-bar* [関数]
戻り値: スクロールバーの相対値
最小値を 0、最大値を 1 としたスクロールバーの示す相対値を取得します。

`make-number-spinner` *value min max step* [*format*] [関数]
戻り値: 数値スピナーコンポーネント
数値スピナーコンポーネントを取得します。min,max を#f にしたときは上限 (下限) がなくなります。format は java.text.DecimalFormat の形式に従います。

`make-tabbed-pane` *component . . .* [関数]
戻り値: タブ領域コンポーネント
タブ領域コンポーネントを生成します。

`get-selected-tab` *tabbed-pane* [関数]
戻り値: GUI コンポーネント
タブ領域コンポーネントにおいて現在選択されているコンポーネントを取得します。

`make-image-icon` *image-file-name* [関数]
戻り値: アイコン
画像アイコンを生成します。

`make-frame-icon` *width height* [*x1 y1 x2 y2*] [関数]
戻り値: アイコン
ペインタにて描画可能なアイコンを生成します。x1,y1,x2,y2 が指定されたときは描画領域の左上を (x1, y1)、右下を (x2, y2) とします。座標は数学等で使用されるものと同じです。

`make-text-pane` [関数]
戻り値: テキスト領域コンポーネント
テキスト領域コンポーネントを生成します。

`set-default-key-action!` *text-pane action . . .* [関数]
戻り値: 未定義
テキスト領域においてキーが押下されたときのデフォルトの動作を指定します。action の指定については make-button-with-action を参照してください。

`add-key-action!` *text-pane key key-modifier action . . .* [関数]
戻り値: 未定義
テキスト領域において指定されたキーが押下されたときの動作を指定します。action の指定については make-button-with-action を参照してください。

`set-style!` *text-pane offset length* [[*attribute-key attribute*] ...] [関数]

戻り値: 未定義
テキスト領域における文字のスタイルを指定します。テキスト領域の offset から length 文字までのテキストが対象になります。

キーワード	内容
font-family	後に続く引数でフォントの種類を指定します
font-size	後に続く引数でフォントサイズを指定します
bold	太文字であることを指定します
italic	斜体であることを指定します
underline	下線を引くことを指定します
strike	取消線を引くことを指定します
superscript	上付き文字を指定します
subscript	下付き文字を指定します
color	後に続く引数で文字の色を指定します
background-color	後に続く引数で背景色を指定します

`beep` [関数]

戻り値: 未定義
ビープ音を鳴らします。

`layout-grid-bag` *panel constraint* [関数]

戻り値: 未定義
指定されたパネルを GridBagLayout に従って整列します。constraint の指定方法は以下の通りです。

```
(((GUI コンポーネント [(属性 . 値) ...])
  ...
 [((GUI コンポーネント [(属性 . 値) ...])
  ...)]
 ...)
```

属性は以下の通りです。GridBag の詳細はこちらを参照してください。

属性	内容
weightx	セルに対するコンポーネントの幅の比率を指定します
weighty	セルに対するコンポーネントの高さの比率を指定します
fill	コンポーネントを拡大するかを指定します。以下のシンボルが指定できます
none	拡大しない
horizontal	水平方向へ拡大する
vertical	垂直方向へ拡大する
both	水平・垂直方向へ拡大する
gridx	x 座標の位置を指定します
gridwidth	x 方向の併合セルを指定します (HTML の<table>タグにおける colspan と同じです

gridheight	y 方向の併合セルを指定します (HTML の<table>タグにおける rowspan と同じです

`layout-grid` *panel layout* [*xgap ygap*] [関数]
戻り値: 未定義
指定されたパネルを GridLayout に従ってマス目状に整列します。並べたいコンポーネントは layout を以下のように並べて指定します。

```
(((1 列目コンポーネント 2 列目コンポーネント ...) ; この列が 1 行目
 ((1 列目コンポーネント 2 列目コンポーネント ...) ; この列が 2 行目
  ...)
```

GridLayout についてはこちらを参照してください。

`layout-border` *panel layout* [*xgap ygap*] [関数]
戻り値: 未定義
指定されたパネルを BorderLayout に従ってマス目状に整列します。並べたいコンポーネントは layout に alist の形で指定します。 alist の値は並べたいコンポーネントを指定します。alist のキーは以下のように指定します。

キーのシンボル	内容
center	値のコンポーネントを中央に配置する
north	値のコンポーネントを上に配置する
south	値のコンポーネントを下に配置する
east	値のコンポーネントを右に配置する
west	値のコンポーネントを左に配置する

BorderLayout についてはこちらを参照してください。

`layout-flow` *component ...* [関数]
戻り値: 未定義
引数のコンポーネントを FlowLayout に従って整列します。FlowLayout についてはこちらを参照してください。

`add-action-listener!` *has-action action-proc* [関数]
戻り値: 未定義
アクションを持つコンポーネントにアクションを追加します。 action-proc はアクションイベントを引数に 1 つ持つ手続きを指定します。

`add-mouse-listener!` *has-mouse-event event-proc* [関数]
戻り値: 未定義
マウスイベントを持つコンポーネントにアクションを追加します。 action-proc はマウスイベントを引数に 1 つ持つ手続きを指定します。

`add-item-listener!` *has-item-event event-proc* [関数]
戻り値: 未定義
アイテムイベントを持つコンポーネントにアクションを追加します。action-proc はアイテムイベントを引数に 1 つ持つ手続きを指定します。

`add-adjustment-listener!` *has-adjustment-event event-proc* [関数]
戻り値: 未定義
調整イベントを持つコンポーネントにアクションを追加します。 action-proc は調整イベントを引数に 1 つ持つ手続きを指定します。

`add-change-listener!` *has-change-event event-proc* [関数]
戻り値: 未定義
変更イベントを持つコンポーネントにアクションを追加します。 action-proc は変更イベントを引数に 1 つ持つ手続きを指定します。

`add-window-listener!` *has-window-event event-proc* [関数]
戻り値: 未定義
ウィンドウイベントを持つコンポーネントにアクションを追加します。 action-proc はウィンドウイベントを引数に 1 つ持つ手続きを指定します。

`get-action-command` *action-event-object* [関数]
戻り値: アクションコマンドの文字列
アクションの手続きの引数で渡されるイベントオブジェクトからアクションのコマンドを取り出します。

`get-when` *action-event-object* [関数]
戻り値: アクションが実行された時間 (SRFI-19 形式)
アクションの手続きの引数で渡されるイベントオブジェクトからアクションが実行された時間を取り出します。

`get-param-string` *action-event-object* [関数]
戻り値: パラメータの文字列
アクションの手続きの引数で渡されるイベントオブジェクトからパラメータの文字列を取り出します。

`get-mouse-button` *mouse-event-object* [関数]
戻り値: 下記参照
マウスイベントの手続きの引数で渡されるイベントオブジェクトからクリックされたマウスボタンを取得します。戻り値は以下の通りです。

ボタンのシンボル	内容
no-button	ボタンはクリックされていません
button1	1 番目のボタンがクリックされました (Windows,Linux 環境では左ボタン)
button2	2 番目のボタンがクリックされました (Windows,Linux 環境では右ボタン)
button3	3 番目のボタンがクリックされました (UNIX ワークステーション等で使用)
unknown-button	値のコンポーネントを左に配置する

`get-mouse-click-count` *mouse-event-object* [関数]
戻り値: クリックされた回数
マウスイベントの手続きの引数で渡されるイベントオブジェクトからクリックされた回数を取得します。

`get-mouse-x` *mouse-event-object* [関数]
戻り値: クリックされたマウス X 座標
マウスイベントの手続きの引数で渡されるイベントオブジェクトからクリックされた場所の X 座標を取得します。座標はシステムの座標系で渡されます。

`get-mouse-y` *mouse-event-object* [関数]
戻り値: クリックされたマウス Y 座標
マウスイベントの手続きの引数で渡されるイベントオブジェクトからクリックされた場所の Y 座標を取得します。座標はシステムの座標系で渡されます。

`popup-trigger?` *mouse-event-object* [関数]
戻り値: ポップアップトリガのとき#t
マウスイベントの手続きの引数で渡されるイベントオブジェクトがポップアップトリガかを調べます。

`get-event-selected-item-value` *item-event-object* [関数]
戻り値: 選択された項目
選択イベントの手続きの引数で渡されるイベントオブジェクトから選択された項目を取得します。

`get-event-adjustment-value` *adjustment-event-object* [関数]
戻り値: 選択された値
調整イベントの手続きの引数で渡されるイベントオブジェクトから選択された値を取得します。

2.13 データベース連携機能

リレーショナルデータベース (RDB) と JDBC により連携する機能です。RDB からのデータを取得するときに RDB の型をメタデータにより参照して、対応する Scheme の型に変換します。

変換の対応表は以下の通りです。

RDB の型	Scheme のオブジェクト
BOOLEAN	#t または#f
CHAR	文字列
DATE	SRFI-19 形式の日付
DECIMAL	正確な数 (有理数)
DOUBLE	不正確な数
FLOAT	不正確な数
INTEGER	正確な整数

NUMERIC	正確な数 (有理数)
REAL	不正確な数
SMALLINT	正確な整数
TIMESTAMP	SRFI-19 形式の日付
TINYINT	正確な整数
VARCHAR	文字列

パラメータは原則として準備された SQL(Prepared SQL) を使用します。Prepared SQL を使用することにより (ドライバが適切に実装されていれば)SQL インジェクション攻撃を防ぐことができます。Scheme オブジェクトをパラメータに渡すときも型変換が適用されます。型の対応は以下の通りです。

Scheme のオブジェクト	RDB の型
文字列	文字列 (String)
短い整数	短い整数 (int)
長い整数	長い整数 (BigInteger)
SRFI-19 形式の日付	日付 (java.sql.Timestamp または java.sql.Date)
不正確な数	浮動小数点型 (double または float)
#t または#f	論理型 (boolean)

クエリの結果は結果バッグ (result-bag) に格納されます。結果バッグからのデータ取得は通常のリストと同様に処理できます。「The Little Schemer」(Daniel P. Friedman, Matthias Felleisen 著 ISBN 978-0-262-56099-3) の「掟 (Commandment)」の書き方にならえば、

> 第一の掟 `(result bag version)`
>
> 結果バッグ `resb` について再帰するときには、次の 2 つの質問を問いかけよ:`(end-result-bag? resb)` と `else`。
>
> 第四の掟 `(result bag version)`
>
> 再帰するとき、少なくとも 1 つの引数はいつも変化する。
> 結果バッグ `resb` について再帰するときには、`(next-result-bag resb)` を使用せよ。

となるとおもいます。

`jdbc-use-db` *dbms-name* [関数]

戻り値: 未定義

使用する DBMS の種類を宣言します。dbms-name には DBMS の種類を表すシンボルを指定します。指定可能なシンボルは以下の通りです。

dbms-name	RDBMS
oracle	Oracle
mysql	MySQL
postgresql	PostgreSQL
sqlite	SQLite

`jdbc-connect` *url* [関数]
戻り値: JDBC コネクション
JDBC を使用して RDBMS に接続します。url は、JDBC で接続するときに指定する URL の文字列です。

`jdbc-prepare` *connection SQL* [関数]
戻り値: JDBC ステートメント
JDBC を使用して SQL を解析し、準備されたステートメントを取得します。

`jdbc-query` *statement param ...* [関数]
戻り値: JDBC 結果バッグ
参照系の JDBC ステートメントを実行します。パラメータはプレースホルダ ("?") の順序に従ってステートメントの後に指定します。

`jdbc-update!` *statement param ...* [関数]
戻り値: 更新された件数
更新系または DDL の JDBC ステートメントを実行します。パラメータはプレースホルダ ("?") の順序に従ってステートメントの後に指定します。

`jdbc-open?` *closable-object* [関数]
戻り値: closable-object が開いていれば#t
closable-object が開いているかを調べます。closable-object には、JDBC コネクション、JDBC ステートメント、JDBC 結果バッグが含まれます。

`jdbc-close` *closable-object* [関数]
戻り値: 未定義
closable-object を閉じます。closable-object には、JDBC コネクション、JDBC ステートメント、JDBC 結果バッグが含まれます。

`get-result-bag-row` *result-bag* [関数]
戻り値: 行
結果バッグの現在の位置から行オブジェクトを取得します。この手続きは通常のリスト処理における car 手続きに相当します。

`get-result-bag-row-list` *result-bag* [関数]
戻り値: 行を表すリスト
結果バッグの現在の位置から行を表すリストを取得します。リストの順序はクエリを実行するときの SQL によって決まります。この手続きは通常のリスト処理における car 手続きに相当します。

`end-result-bag?` *result-bag* [関数]
戻り値: 結果バッグが行の終端を超えたら#t
結果バッグが行の終端を超えているかを調べます。この手続きは通常のリスト処理における null?手続きに相当します。

`next-result-bag` *result-bag* [関数]
戻り値: 結果バッグ
結果バッグの「カーソル」を次の行に進めます。この手続きは通常のリスト処理における cdr 手続きに相当します。

`row-ref` *row-of-result-bag symbol-of-row-id* [関数]
戻り値: 行の要素に対応するオブジェクト
結果バッグの行オブジェクトから行に対応するオブジェクトを取得します。第 1 引数には結果バッグの行を、第 2 引数には行名を表すシンボルを指定します。

2.14 サウンド機能

サウンドデバイスを操作する機能です。この機能を使用するには (use sound) が必要です。

`make-audio-format` *encoding sample-rate sample-size-in-bits channels frame-size frame-rate endianness* [関数]
戻り値: オーディオフォーマットオブジェクト
オーディオフォーマットオブジェクトを取得します。引数の意味は以下の通りです。

encoding — エンコードの種類をシンボル指定します。指定できるシンボルは以下の通りです。
- alaw: a-law
- pcm-signed: 符号付きリニア PCM
- pcm-unsigned: 符号なしリニア PCM
- ulaw: u-law

sample-rate — サンプリング周波数を実数で指定します。
sample-size-in-bits — 各サンプルのビット数を指定します。
channels — チャネル数を指定します。モノラルならば 1、ステレオならば 2 となります。
frame-size — 各フレームのバイト数を指定します。
frame-rate — 1 秒あたりのフレーム数を指定します。
endianness — エンディアンを指定します。ビッグエンディアンならばシンボル big-endian、リトルエンディアンならばシンボル little-endian を指定します。

`make-target-data-line` *audio-format* [関数]
戻り値: ターゲットデータラインオブジェクト
ターゲットデータライン (入力) を生成します。

`make-source-data-line` *audio-format* [関数]
戻り値: ソースデータラインオブジェクト
ソースデータライン (出力) を生成します。

`open-data-line` *data-line* [関数]
戻り値: 未定義
データラインをオープンします。

`start-data-line` *data-line* [関数]
戻り値: 未定義
データラインを開始します。このことによりデータがデバイスより入出力されます。

`stop-data-line` *data-line* [関数]
戻り値: 未定義
データラインを停止します。

`flush-data-line` *data-line* [関数]
戻り値: 未定義
データラインにあるデータをクリアします。

`close-data-line` *data-line* [関数]
戻り値: 未定義
データラインをクローズします。

`read-data-line-relative` *target-data-line sample-num* [関数]
戻り値: 取得したデータの f32 一様ベクトル
ターゲットデータラインからデータを読み込みます。強度は最大ビット数で規格化され、0〜1.0(符号なし) または-1.0〜1.0(符号付き) になります。
結果は f32 一様ベクトル (SRFI-4 参照) として返ります。
データは入力チャネル数だけの多値として得られます。(モノラルならば 1 値、ステレオならば 2 値)

`write-data-line-relative` *source-data-line data-to-write* [関数]
`[...]`
戻り値: 書き込まれたデータ数
ソースデータラインにデータを書き込みます。強度は最大ビット数で規格化され、0〜1.0(符号なし) または-1.0〜1.0(符号付き) になります。
データは f32 一様ベクトル (SRFI-4 参照) として与えます。
データは入力チャネル数だけ与えます。(モノラルならば 1 つ、ステレオならば 2 つ)

`make-playable` *sound-file-name repeat* [関数]
戻り値: 演奏オブジェクト
サウンドファイルを演奏するオブジェクトを取得します。サウンドファイルのフォーマットは使用している Java の環境に依存します。Java 標準では WAV ファイルが使用できます。MP3 プラグインを使用することで MP3 にも対応できます。repeat は繰り返して演奏する回数を指定します。#f のときは無限に繰り返します。

`play-sound` *playable* [関数]
戻り値: 演奏が開始されれば#t
指定した演奏オブジェクトの演奏を開始します。

`stop-play-sound` *playable* [関数]
戻り値: 演奏が終了されれば#t
指定した演奏オブジェクトの演奏を終了します。

`pause-play-sound` *playable* [関数]
戻り値: 演奏が一時停止されれば#t
指定した演奏オブジェクトの演奏を一時停止します。再開するには play-sound を再び呼び出します。

2.15 プラグイン機能

Schluessel プラグインを読み込む機能です。

`*plugin-path*` [変数]
load-plugin にてプラグイン JAR を検索するときのディレクトリのパスが格納されている変数です。変更するときは専用の手続き load-use-path を使用してください。

`add-plugin-path` ディレクトリ名 *endflg* [関数]
戻り値: 未定義
*plugin-path*にパッケージパスを追加するときに使用する手続き式です。endflg に#t を指定すると*plugin-path*の末尾にパスが追加されます。その他の時は*load-path*の先頭に追加されます。

2.16 その他の関数

`apply-java` *java-object java-method-name* [*args . . .*] [関数]
戻り値: Java メソッドの戻り値
Java オブジェクトのメソッドを呼び出すことができます。メソッド名はシンボルで与えます。Java オブジェクトでなくても、GUI コンポーネントなどに適用すると、ラップされている Swing のオブジェクトのメソッド (プロパティ) を呼び出すことができます。戻り値は可能な限り Scheme のオブジェクトに変換されます。

`apply-java-static` *class-name java-method-name* [*args . . .*] [関数]
戻り値: Java メソッドの戻り値
Java オブジェクトの static メソッドを呼び出すことができます。クラス名とメソッド名はシンボルで与えます。

`get-system-property` *prop-name* [関数]
戻り値: Java システムプロパティ; ないときは#t
Java のシステムプロパティを取得します。

`get-system-properties` [関数]
戻り値: Java システムプロパティの alist
Java のシステムプロパティを取得します。

`set-system-property!` *prop-name prop* [関数]
戻り値: 未定義
Java のシステムプロパティを設定します。

`tr-string` *src-string old new* [関数]
戻り値: 変換された文字列
文字列を変換します。この手続きは UNIX の tr コマンドのように動作します。

`quasiquote-string` *string* [関数]
戻り値: 評価された文字列
文字列内に,(*string* ...) で表現された S 式を評価して、結果の文字列と結合します。

`promise?` *object* [関数]
戻り値: 約束であれば#t
delay で生成された約束であるかを調べます。

`define-feature` *feature-name* [*body* ...] [特殊フォーム]
戻り値: 未定義
SRFI-0 で使用される機能を定義します。

2.17 Applet 連携機能

Scheme プログラムを Applet として動作させることができます。Applet の動作方法は HTML ファイルに以下の記述を付け加えます。

```
<applet
 codebase="."
 code="net.morilib.lisp.applet.SchlushApplet.class"
 archive="schluessel.jar"
 width="400" height="200">
 <param name="scheme-program"
  value="Scheme プログラム.scm" />
</applet>
```

`*applet*` [変数]
Applet をグラフィック領域のフレームを表す変数です。

2.18 Servlet 連携機能

Servlet と連携して、Scheme にてサーバサイド Java を利用した Web アプリケーションを記述することができます。

サーバサイド Schluessel を使用するためには Tomcat 等の Web コンテナが必要です。Web コンテナの設定方法は次の通りです。

1. Web コンテナのルート配下に WEB-INF ディレクトリとその配下に lib, classes というディレクトリを用意します。

2. lib ディレクトリ配下に schluessel.jar と schluessel-sss.jar を配置します。
3. WEB-INF ディレクトリ配下に web.xml というファイルを生成し、次のように生成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:web=
  "http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
 id="WebApp_ID"
 version="2.5">
  <servlet>
    <servlet-name>Schluessel</servlet-name>
    <servlet-class>
 net.morilib.lisp.sss.servlet.http.SchlushHttpServlet
    </servlet-class>
    <init-param>
      <param-name>scheme-root</param-name>
<!-- ↓ Scheme ソースがあるパッケージのルートを記述する -->
      <param-value>test.scheme</param-value>
    </init-param>
    <init-param>
      <param-name>scheme-encoding</param-name>
      <!-- ↓ Scheme ソースのエンコードを記述する -->
      <param-value>UTF-8</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Schluessel</servlet-name>
    <url-pattern>*.scm</url-pattern>
  </servlet-mapping>
</web-app>
```

4. web.xml にて指定した Scheme の Web アプリケーションがあるパッケージに Scheme ソースを配置します。

このとき、Web コンテナは次のようになっていると思います。(ここでは、ルートパッケージを test.scheme としています)

```
Web コンテナのルート
 +- WEB-INF
     + lib
     + classes
     |  + test
     |     + scheme
```

```
|         + Scheme ソース (拡張子は.scm)
+ web.xml
```

配置したアプリケーションは

`http://ホスト名:nnnn/Web コンテナ名/WebApplicationName.scm`

で参照できます。

Web アプリケーションと使用する Scheme アプリケーションは

- do-get (HTTP GET メソッドを処理)
- do-post (HTTP POST メソッドを処理)
- do-delete (HTTP DELETE メソッドを処理)
- do-head (HTTP HEAD メソッドを処理)
- do-options (HTTP OPTIONS メソッドを処理)
- do-put (HTTP PUT メソッドを処理)
- do-trace (HTTP TRACE メソッドを処理)

の手続きのいずれかを定義することが必要です。これらの手続きは 2 つの引数

- HTTP リクエストオブジェクト
- HTTP レスポンスオブジェクト

を伴って呼び出されます。これらのオブジェクトの詳細については以下を参照してください。

`get-parameter` *request string-of-param-name* [関数]
戻り値: パラメータを表す文字列
HTTP リクエストオブジェクトから文字列でパラメータを受け取ります。

`get-parameter-integer` *request string-of-param-name* [関数]
戻り値: パラメータを表す整数
HTTP リクエストオブジェクトから整数でパラメータを受け取ります。パラメータが整数に解析できなかったときはエラーを raise します。

`set-character-encoding` *request encoding* [関数]
戻り値: 未定義
フォームの文字コードを指定します。適切な文字コードを指定しなかったときは get-parameter の結果が文字化けすることがあります。

`session-valid?` *request* [関数]
戻り値: セッションが有効ならば#t
現在のセッションが有効かを調べます。

`get-session` *request* [関数]
戻り値: HTTP セッションオブジェクト
HTTP セッションを取得します。現在のセッションが無効なときは新規に生成します。

`set-content-type` *response content-type* [関数]
戻り値: 未定義
Content-Type をセットします。

`get-output-port` *response* [関数]
戻り値: 出力ポート
レスポンスを送る出力ポートを取得します。

`get-attribute` *attribute-holder string-of-attr-name* [関数]
戻り値: パラメータを表す整数
属性を属性ホルダーから取得します。「属性ホルダー」とは、HTTP リクエストオブジェクト、HTTP セッション、Servlet コンテキストを指します。

`set-attribute!` *attribute-holder string-of-attr-name datum* [関数]
戻り値: パラメータを表す整数
属性を属性ホルダーにセットします。「属性ホルダー」とは、HTTP リクエストオブジェクト、HTTP セッション、Servlet コンテキストを指します。
任意の Scheme オブジェクトをセットすることができます

索引

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W

X

Schluessel ユーザリファレンス
2011 年 12 月 31 日発行

サークル: morilib.net
発行人: 森口 雄一郎 (代官山 唯一)
印刷所: しまや出版様