NAL

プリセットクラス
リファレンス

はじめに

知的財産権について

アクト株式会社

改訂履歴

1999/08/05	初版
1999/09/14	Stream クラス追加
1999/10/02	Class.assign(OBJECT,CLASS)メソッドを追加
1999/11/17	Socket クラス追加
2000/01/20	Date クラスにコンストラクタ Date(実数)を追加
2000/05/27	Array.purge()に重複要素削除オプションを追加
2000/09/02	Buffer クラスに encode64()、decode64()を追加
2000/09/15	DBMAN クラスの put()メソッド引数に Date オブジェクトを追加
2000/09/19	Socket クラスのプログラミング例中の記述間違いを訂正
2000/10/01	Object.toHash()クラスメソッドを追加
2000/12/07	document クラスを追加
2001/06/03	DBMAN クラスに setIndex()、matchOf()メソッド追加
2001/07/09	Buffer クラスに CRC()メソッド追加
2001/08/20	System クラスを追加
2002/01/06	Date クラスに addYear()..addSeconds()を追加
2002/04/09	DBMAN クラスに recordset()メソッドを追加
2002/12/18	Storage クラスに time()メソッドを追加
2003/02/06	Wrapper クラスを追加
2003/04/29	System クラスに worning()メソッドを追加
2003/06/24	Object クラスの toHash()メソッドの仕様を拡張
2003/06/30	Object クラスに getIdent()メソッドを追加
2003/07/03	Math クラスに gcd(),prime(),odd()メソッドを追加
2003/11/08	Zip クラスを追加
2004/03/21	Buffer クラスに getText()メソッドを追加
2004/05/21	Buffer クラスに Unicode 対応メソッドを追加
2004/06/16	Array.sort()の比較関数の返す値に論理値を許可
2005/06/15	Buffer クラスの put 系インスタンス・メソッド　リターン値を this に変更
2005/07/11	Buffer クラスに search()メソッドを追加
2006/12/16	DBMAN クラスに recordset(callback:method):DBREC[] メソッドを追加

1 null クラス

null 値に対するメソッドを提供する隠蔽されたクラス

メソッド	
getName():string	"null"を返す
getValue():null	null を返す
toString():string	"null"を返す

2 int クラス

プリミティブ型である整数値に対するメソッドを提供する隠蔽されたクラス

メソッド	
getName():string	"int"を返す
getValue():int	この値そのものを返す
isInt():bool	true を返す
toIntValue():int	値をそのまま返す
toRealValue():real	実数化した値を返す 例： (100).toRealValue() ⇒ 100.0
toChar():string	下位 16 ビットにつき、その値に相当する文字を文字列として返す。 8 ビットを超える場合は、第 1 バイトに上位 8 ビットを、第 2 バイトに下位 8 ビットを格納する。 例えば、“ 亜 ”はシフトＪＩＳコードで 0x889F、ＥＵＣコードで 0xB0A1 と規定されている。従って 0x889F.toChar()はシフトＪＩＳ文字“ 亜 ”となり、0xB0A1.toChar()はＥＵＣ文字“ 亜 ”となる。
isEven():bool	偶数の場合のみ true を返す
toString():string	10 進数表記に文字列化した結果を返す。
toString(format:string):string	format に指定する、string クラスの import()仕様書式文字列に従って文字列化する。
toString(radix:int=10, fig:int=0, term:string=","):string	radix 進数表記に文字列化した結果を返す。(2 ≦ radix ≦ 36) fig に 1 以上の整数を指定すると、fig 桁ごとに term を挿入する。 （10 進数文字列の場合は -2147483648～2147483647 まで）
getLittle():real	0.0 を返す

3 real クラス

プリミティブ型である実数値に対するメソッドを提供する隠蔽されたクラス

メソッド	
getName():string	"real"を返す
getValue():real	この値そのものを返す
isInt():bool	false を返す
toIntValue():int	整数化した値を返す（少数部切り捨て） 例： (3.14).toIntValue() ⇒ 3
toRealValue():real	値をそのまま返す
toChar():string	整数化してその下位 16 ビットにつき、その値に相当する文字を文字列として返す。 8 ビットを超える場合は、第 1 バイトに上位 8 ビットを、第 2 バイトに下位 8 ビットを格納する。 例えば、“ 亜 ”はシフトＪＩＳコードで 0x889F、ＥＵＣコードで 0xB0A1 と規定されている。従って 0x889F.toChar()はシフトＪＩＳ文字“ 亜 ”となり、0xB0A1.toChar()はＥＵＣ文字“ 亜 ”となる。
isEven():bool	false を返す
toString(format:string=""):string	10 進表記に文字列化した結果を返す。format に string クラスの import()仕様書式文字列を指定した場合はその指定に従って文字列化する。
getLittle():real	小数部を返す。 例： (3.14).getLittle() ⇒ 0.14

4 number クラス

プリミティブ型である整数値、実数値に対するメソッドを提供する隠蔽されたクラス

メソッド	
getName():string	整数なら"int"を、実数なら"real"を返す
getValue():number	この値そのものを返す
isInt():bool	整数なら true を返す
toIntValue():int	実数の場合、整数化した値を返す（少数部切り捨て） 例： (3.14).toIntValue() ⇒ 3
toRealValue():real	整数の場合、実数化した値を返す 例： (100).toRealValue() ⇒ 100.0
toChar():string	実数の場合は整数化し、下位 16 ビットにつき、その値に相当する文字を文字列として返す。 8 ビットを超える場合は、第 1 バイトに上位 8 ビットを、第 2 バイトに下位 8 ビットを格納する。 例えば、“亜”はシフトＪＩＳコードで 0x889F、ＥＵＣコードで 0xB0A1 と規定されている。従って 0x889F.toChar()はシフトＪＩＳ文字“亜”となり、0xB0A1.toChar()はＥＵＣ文字“亜”となる。
isEven():bool	整数かつ偶数の場合のみ true を返す
toString():string	整数か実数に応じ、各々の toString()メソッド結果の文字列を返す。
getLittle():real	値が整数なら 0.0 を、実数なら小数部の値を返す。

5 bool クラス

プリミティブ型である論理値に対するメソッドを提供する隠蔽されたクラス

メソッド	
getName():string	"bool"を返す
getValue():int	この値そのものを返す
toIntValue():int	true は 1、false は 0 を返す 例： true.toIntValue() ⇒ 1
toString():string	true は"true"、false は"false"とした文字列を返す

6 string クラス

プリミティブ型である文字列値に対するメソッドを提供する隠蔽されたクラス

※以下「位置」とは表記上の１文字を１と数える文字数による文字列先頭からの文字インデックスを意味し、先頭文字は０である。

メソッド	
charAt(index:int=0):string	index 文字位置の１文字を返す。
charCodeAt(index:int=0):int	index 文字位置の文字コードを整数値として返す。 該当文字がマルチバイトの場合は(第１バイト×256＋第２バイト)値を返す。
chomp(str:string="\r\n"):string chop(str:string="\r\n"):string	最後から str に含むいずれかの文字が連続する場合、これを取り除いた文字列を返す。str の指定が無い場合は改行文字（¥r,¥n）を指定した場合と同じ。
eval():variant	文字列を、式の表記として解析した結果を返す。
extract(lead:string="", trail:string="", start:int=0):string	start 文字位置以降で lead と trail で挟まれた文字列を抽出する。結果に lead および trail は含まない。 lead に""を指定した場合は start 位置から、trail に""を指定した場合は最後までを抽出する。 lead が含まれない場合は""を返し、lead 以降に trail が含まれない場合は最後までを抽出する。
extractNum(start:int=0):string	start 文字位置以降で数表記とみなせる文字列を抽出する。なければ空文字列を返す。 数表記とは半角数字あるいはそれが連続し、最初の"."は後続数字があれば数表記に含め、また"+"あるいは"-"は数表記を先導する場合には数表記に含める。 例："経済成長率は+3.47%です".extractNum() ⇒ "+3.47"
getName():string	〝string〟を返す
getValue():string	この値そのものを返す
import(val:number or string,,) :string	文字列で示す書式フォーマットに従って、val 列で示す値を順に文字列内に取り込んだ結果の文字列を返す。値として論理値、null、オブジェクトを指定した場合は書式フォーマットは機能せず、値を文字列化した結果を取り込む。 例：〝今日は%2d 月%02d 日です〟.import(4,1); の結果は、〝今日は 4 月 01 日です〟を返す。 ※ 書式フォーマットについては後述する ※ 結果文字列長は 1M バイトを越えない。
includes(key:string):bool	文字列式 key を含んでいれば true を返す。indexOf()参照
includes(keys[]:string):bool	keys に示す文字列配列の要素のうちどれかを含んでいれば true を返す。
isKcode():int	文字列を構成する漢字コードを調査し、結果を返す。 0==漢字を含まない、1==シフトＪＩＳ、2==ＥＵＣ、3==ＪＩＳ、4==シフトＪＩＳかＥＵＣか不明
indexOf(key:string,start:int=0) :int	start 以降に文字列式 key を含む場合、その開始位置返す。結果の値 0 は先頭を意味し含まない場合は-1 を返す。start を指定しない場合は先頭から検索する。

indexOf(keys[]:string,start:int=0) :int	start 以降に keys に示す文字列配列要素のうちどれかを含む場合、そのうちのもっとも start 寄りにある開始位置を返す。結果の値 0 は先頭を意味し含まない場合は-1 を返す。 start を指定しない場合は先頭から検索する。
lastIndexOf(key:string, end:int=length()):int	end 以前に文字列式 key を含む場合、その開始位置を返す。結果の値 0 は先頭を意味し含まない場合は-1 を返す。end を指定しない場合は最後の文字から検索する。
lastIndexOf(keys[]:string, end:int=length()):int	end 以前に keys で示す文字列配列要素のうちどれかを含む場合、そのうちのもっとも後ろにある開始位置を返す。結果の値 0 は先頭を意味する。含まない場合は-1 を返す。 end を指定しない場合は最後の文字から検索する。
matchOf(key:string,start:int=0) :int	start 以降に文字列式 key 中の何れかの文字を含む場合、その位置を返す。結果の値 0 は先頭を意味し含まない場合は-1 を返す。start を指定しない場合は先頭から検索する。 key に任意組数の半角文字範囲指定（"0..9"、"a..z"、"0-9"、"a-z"など）を含めてもよい。
lastMatchOf(key:string, end:int=null):int	end 以前に文字列式 key 中の何れかの文字を含む場合、その位置を返す。結果の値０は先頭を意味し含まない場合は-1 を返す。end を指定しない場合は、最後の文字から検索する。 key に任意組数の半角文字範囲指定（"0..9"、"a..z"、"0-9"、"a-z"など）を含めてもよい。
search(keys[]:string, start:int=0):int	start 以降に keys に示す文字列配列の要素のうちどれかを含む場合、そのうちのもっとも start 寄りにある配列要素の番号を返し、含まない場合は-1 を返す。
length():int length:int	文字列の長さを表記文字構成数として返す。すなわちマルチバイト文字も１文字として扱う。 例："漢字 abc".length() ⇒ 5
byteLength():int	文字列のバイト長を返す。　例："漢字 abc".byteLength() ⇒ 7
parseInt():int	先頭文字から整数表記とみなせる範囲までを整数として評価してその値を返す。整数とみなせる文字がない場合は 0 を返す。
parseReal():real parseFloat():real	先頭文字から実数表記とみなせる範囲までを実数として評価してその値を返す。実数とみなせる文字がない場合は 0.0 を返す。
parseLiteral():string	値が≪文字列定数記述≫（string literal）である文字列を解釈した結果を返す。 値が≪文字列定数記述≫でない場合は文字列をそのまま返す。
parseEntity():string	実体参照表現を文字に変換した結果の文字列を返す。置換対象および置換結果は "& "→"&"、"<"→"<"、">"→">"、"""→'"'、" "→" "[1]、"'"→"'" に置換し、"&#<decimal>;" および "&#x<hex>;" は 10 進表記<decimal>、16 進表記<hex>に対応する Uni-Code 文字を Shift-JIS 文字に変換して置換する。（※toEntity()の逆機能）
parseString(start:int=0):string	start 位置以降の≪文字列定数記述≫を取り出す。 文字列定数記述がなければ空文字列を返す。（※toString()の逆機能）
replace(str:string,to:string) :string	文字列中の str 文字列パターンを to 文字列パターンにすべて置き換えた結果を返す。 str を含まない場合は、同じ値を返す。

[1] RFC では " " は "¥xA0" であるが、toEntity()との互換のため "¥x20" に置換する。

reverse():string	文字並びを反転した結果を返す。 ("ABC" ⇒ "CBA")
split(term:string,cont:bool=false) :string[]	term で指定する文字列を区切り文字列として、分割した文字列配列として返す。 文字列中に term を含まない場合は、文字列全体を単一要素とする配列を返す。 例："今日の天気は晴れです".split("天気") ⇒ {"今日の","は晴れです"} となる。 cont に true を指定した場合、連続する term 値を１つの term 値として扱う。 例:str.split(" ",true)は連続するスペース文字を１つのスペース文字として扱う。
substr(start:int=0,cnt:int=null) :string	start 位置から cnt 文字分の文字列を抽出して返す。cnt を指定しない場合は最後までを返す。
substr(start:string,cnt:int=null) :string	最初に検出される start 文字列パターン直後の、cnt 文字分の文字列を抽出して返す。cnt を指定しない場合は最後までを返す。start パターンがなければ先頭からとなる。
substring(start:int=0, end:int=null):string	start 位置から end 位置までの文字列を抽出して返す。end を指定しない場合は最後までを返す。 end に負値を指定した場合は 0 として扱う。
substring(start:int=0, end:string):string	start 位置以降 end 文字列パターンまでの範囲を抽出する。end に指定する文字列を含んだ結果となる。
substring(start:string, end:string,opt:int=0):string	start 文字列パターン以降 end 文字列パターンまでの範囲を抽出する。opt に指定する整数値によって以下の機能となる。 0 は start および end に指定する文字列を含んだ結果となる。 1 は start を含まず end を含む。 2 は start を含み end を含まない。 3 は start および end を含まない。※この場合は extract()と同じ
trim(str:string=" "):string	先頭および末尾に str に含むいずれかの文字が連続する場合、これを取り除いた文字列を返す。 str の指定が無い場合は半角スペース文字を指定した場合と同じ。
toUpperCase():string	半角英字を大文字化した結果を返す。
toLowerCase():string	半角英字を小文字化した結果を返す。
toSJIS():string	ＥＵＣまたはＪＩＳ文字列をシフトＪＩＳ文字列に変換した結果を返す。（※半角カナ未対応）
toEUC():string	シフトＪＩＳ文字列をＥＵＣ文字列に変換した結果を返す。
toJIS(KI:string,KO:string):string	シフトＪＩＳ文字列をＪＩＳ文字列に変換した結果を返す。半角カナは全角カナに変換する。 KI には漢字シフトイン文字列を、KO には漢字シフトアウト文字列を指定するが、各々指定しない場合は KI に"¥x1B$B"を、KO に"¥x1B(B"を指定した場合と同じ結果を返す。
toEntity():string	実体参照表現に変換した結果を返す。実体参照表現への置換対象文字および置換結果は "&"→"&"、"<"→"<"、">"→">"、'"'→"""、"'"→"'"[2]、"\r"(復帰)→"
"、 "\n"(改行)→"
"、" "→" "[3] ※逆変換は parseEntity()

[2] XML では "'" となるべきであるが、HTML においては定義がないため
[3] RFC では " " は "¥xA0" であるが、HTML において<PRE></PRE>外で半角スペースを表現したい場合が多いことを勘案して "¥x20" を " "に置き換える。

toString():string	≪文字列定数記述≫に変換した結果を返す。 ≪文字列定数記述≫とは、文字列全体を“〃”（ダブルクォーテーション）で括り、“〃”は“¥〃”にタブコードは“¥t”に、復帰改行文字は“¥r”、“¥n”に“¥”文字は“¥¥”に、0x01 から 0x1F の間のコードは“¥x01”、“¥x1F”のように１６進文字表記に変換する。 ※逆変換は parseString()
toHexPtn(spc:bool=false):string	文字列を構成するバイト列を HEX 文字に変換した結果の文字列を返す。 例："ABC" ⇒ "414243" "漢字" ⇒ "8ABF8E9A" spc に true を指定した場合は 1 バイト分ごとに半角スペースを埋め込む。
MD5():string	MD5 メッセージダイジェスト化した 32 文字の HEX 文字列を返す。

6.1 import(argument,,,)メソッドに関するフォーマット書式

書式文字列.import(argument,,,)メソッドにおける書式指定は、次の形式で行う。

%[<flags>] [<width>] [.<precision>] <type>

各書式指定フィールドでは、特定の書式を表す 1 個の文字または数字を指定する。
書式文字列中の<type>文字が下記に該当しない場合は、パーセント記号を含めて次のパーセント記号直前までの文字列を出力する。
但し %; はパーセント記号のみを出力する。
オプションのフィールド群は type 文字の前に指定する。
例：

```
var str:string = 〝今日は%d 日です〞.import(new Date().getDate());
```

■<type>
下表に示す半角英字１文字によって argument のデータ型と出力書式を指定する。※データ型が不正な場合は誤動作する。

文字	データ型	出力書式
d、i	int	符号付き 10 進整数
u	int	符号なし 10 進整数
o	int	符号なし 8 進整数
x、X	int	符号なし 16 進整数、x は 10～15 の表現に"abcdef" を使用し、X は"ABCDEF" を使用。
e、E	real	[-]d.dddd e [sign]ddd 形式符号付きの値。d は 1 個の 10 進数、dddd は 1 個または複数の 10 進数、ddd は正確に 3 桁の 10 進数、sign は + または -。

		E は、指数の前に付くのが e ではなく E である点を除いて、e の書式と同じ。
f	real	[-]dddd.dddd 形式の符号付きの値。dddd は、1 個または複数の 10 進数。小数点の前の桁数はその数の絶対値によって決定され、小数点の後の桁数は要求された精度によって決定される。
g、G	real	f または e の書式のうち、指定された値および精度を表現できる短い方の書式。e 書式は、値の指数部が、-4 より小さいか precision で指定された数よりも大きい場合にのみ使う。後に続く 0 は切り捨てられ、小数点は 1 個または複数の数字が続く場合にのみ表示される。 G は、指数の前にあるのが e ではなく E である点を除いて、g の書式と同じ（該当する場合）。
s、S	variant	値を既定書式で文字列化したものが precision の範囲まで表示される。

※符号なし整数を扱わないため、負の整数値に対する符号なし表示（u、o、x、X）は実体を表していない。

■<flags>

下表に示す半角記号によって出力の位置決めと符号、空白、小数点、8 進数と 16 進数のプレフィックスの出力を制御する。1 つの書式指定に、複数のフラグを指定できる。

フラグ	意味	デフォルト
-	指定されたフィールド幅に結果を左詰めする。	右詰め
+	出力値が符号付きの場合は、出力値の前に + または - の符号を付ける。	負の符号付きの値にだけ符号 (-) を表示。
0	width の前に 0 を付けると、最小幅まで 0 が付加される。0 と - を指定すると、0 は無視される。	パディングなし。
空白 (' ')	出力値が符号付きで整数であると、出力値の前に空白が付く。空白フラグと + フラグの両方を指定すると、空白フラグは無視される。	空白は表示されない。
#	o、x、X の各書式と一緒に使うと、0 以外のすべての出力値の前にそれぞれ 0、0x、0X が付く。	空白は表示されない。
	e、E、f の各書式と一緒に使うと、必ず出力値に強制的に小数点が入る。	後続の数値がある場合にのみ小数点が表示される。
	g または G の書式と一緒に使うと、必ず出力値に強制的に小数点が入り、後続する 0 が切り捨てられない。d、i、u、s のいずれかと一緒に使うと無視される。	後続の数値がある場合にのみ小数点が表示される。後続の 0 は切り捨てられる。

■<width>
負でない 10 進整数を指定して、表示する最小文字数を制御する。
出力値の文字数が指定幅より少ない場合は、- フラグ（左詰め）指定の有無によって、値の左または右に最小幅まで空白が追加される。width の前に 0 を付けると、最小幅まで 0 が追加されます（左詰めの数には無効）。
文字幅指定では、値は切り捨てられない。出力値の文字数が指定文字幅より多いか、または width を指定しないと、値のすべての文字が精度の指定に従って表示される。
文字幅指定にアスタリスク（*）を指定すると、import()メソッドの引数リスト中の対応する int 引数から値が取られる。width は、引数リストの中で、書式化する値の前に指定する。幅も指定しないか、必要なサイズよりも小さい幅を指定しても、フィールドは切り捨てられず、フィールドを変換結果に合わせて拡張する。

■<precision>
ピリオド記号に続く負でない 10 進整数を指定してオプションの数字出力フィールドの全体または一部に表示する最大文字数、または整数値として表示する最小桁数を指定する 。

type 文字	意味	デフォルト
d、i、u、o、x、X	表示する最小限の桁数を指定する。引数中の桁数が precision より小さい場合は、出力値の左に 0 が充填される。桁数が precision を超えた場合でも、値は切り捨てられない。	デフォルトの精度は 1。
e、E	小数点以下の表示桁数を指定する。表示される最後の数字は丸められる。	精度のデフォルト値は 6。precision を 0 に指定するか、後続する数字を指定しないでピリオド (.) を指定すると、小数点は表示されない。
f	小数点以下の桁数を指定する。小数点が表示される場合は、その前に少なくとも数字が 1 つ表示される。値は適切な桁数まで丸められる。	精度のデフォルト値は 6。precision を 0 に指定するか、後続する数字を指定しないでピリオド (.) を指定すると、小数点は表示されない。
g、G	表示する最大有効桁数を指定する。	6 桁の有効桁が表示され、後続の 0 は切り捨てられる。
s、S	表示する最大文字数を指定する。precision を超えた文字は表示されない。	すべて表示される。

7 array クラス

配列を扱うメソッドを提供する抽象クラスであり、Object クラスと Array クラスはこのクラスを includes している。
ユーザクラスにて、このクラスを拡張することも、includes することもできない。
提供されるインスタンス・メソッドについては Array クラスを参照のこと。

8 Value クラス

値を扱うクラスメソッドを提供する抽象クラス。クラス拡張はできない。

クラス・メソッド	
typeName(val:variant):string	val 値の型名を返す。型名については typeof() 演算子の解説を参照
toString(val:variant):string	val 値を文字列表記に変換して返す
toJSON(val:variant):string	val 値を JSON 書式文字列にして返す ※4
serialize(val:variant):string	val 値をシリアライズ化した文字列を返す。 シリアライズ化した文字列から実体化するには Value.eval()を利用する。
eval(src:string):variant	src に与えられた、Value.serialize()で生成したシリアライズ化した文字列を解釈して返す
pack(val:variant, encodeMethod:bool=true) :string	val 値をパック文字列に変換して返す。 encodeMethod に true を指定すると、val がインスタンスの場合にそのインスタンス・メソッドを含めてパック化する。
unpack(src:string):variant	src に与えられた、Value.pack()で生成したパック化した文字列を解釈して返す

8.1 toJSON()の使用例

toJSON()は、インスタンスあるいはハッシュ、配列といった構造化されたデータをクライアント・サイドの JavaScript を使って処理する場合に、JavaScript で解釈可能な形式である JSON フォーマットに変換した文字列を生成する。
例：

```
var     obj = object:{
        field   data:string = "データ";
        };
<INPUT type="text" id="DSP">;
<SCRIPT><!-- 以下は JavaScript ソース
var json = %{ Value.toJSON(obj) }%;
document.getElementById("DSP").value=json.data;
//-->;
</SCRIPT>
```

8.2 Value.serialize()、Value.eval()の使用例

代表的な使用方法としては、文字ベースの通信をおこなうノード間で、クラス定義と共に値を引き渡す場合などに使用する。

[4] JSON は（JavaScript Object Notation）の略

例：

```
class   Agent(data:string){
        public  field   clsPack:string = Value.serialize(Object.getClass(this)); // 自身のクラスをシリアライズ化して保持
        public  method  view(){ writeln("data=",data); }
        }
var     agent:Agent = new Agent("Hello");               // “Hello”を引数として Agent インスタンスを生成
var     packs:string = Value.serialize(agent);          // インスタンスをシリアライズ化
//ここで文字列 packs を他のノードに通信で渡すなどする

//以下は受け取ったノードでのプログラム例
var     rcvs:object = Value.eval(packs);                // シリアライズ化されたインスタンスをハッシュとして復元
var     clsdef:class = Value.eval(rcvs.clsPack);        // シリアライズ化された Agent クラス定義を復元
var     instance:Agent = Class.assign(rcvs,clsdef);     // Agent インスタンスとして構築して返す
instance.view();                                        // Agent クラスのインスタンスとして振る舞う
```

9 Class クラス

クラスに関するクラスメソッドのみ提供する抽象クラス。

クラス・メソッド	
getSuper(cls:class):class	cls で示すクラスの直接基本クラスを得る。
assign(obj:object, cls:class=null):object	obj で示すオブジェクトまたはハッシュを cls で示すクラスにアサインし、obj への参照を返す。以後、obj に対して cls クラスのメソッドを使用できる。cls を指定しない場合は obj の持つ同名のクラスを指定したものとみなす。 正しくない cls を指定した場合は null を返す。 ※組み込みクラスにアサインしてはならない
toHash(obj:object):object	obj オブジェクトおよび obj が参照するすべてのオブジェクトをハッシュ化し、obj への参照を返す。
toInstance(obj:object, cls:class=null):object	obj で示すハッシュ値を cls で示すクラスのインスタンスに取り込み、そのインスタンスへの参照を返す。cls を指定しない場合は、obj が持つ同名のクラスを指定したものとみなす。クラスが存在しない場合は null を返す。

9.1 Class.assign()の使用例

```
class   Agent(data:string){
        field   clsPkg:string;
        method  view(){writeln("data=",data);}
        }
var     obj:Agent = new Agent("Hello");         // Agent インスタンスを生成
obj.clsPkg = Value.serialize(Agent);            // Agent クラスをシリアライズ化してオブジェクト内に格納
var     packs:string = Value.serialize(obj);    // Agent インスタンスをシリアライズ化
// ※ここで packs を他のノードに渡す

以下は受け取ったノードでのプログラム例
var     rcv:object = Value.eval(packs);         // シリアライズ化した文字列からインスタンスを復元（ハッシュとなる）
var     cls:class = Value.eval(rcv.clsPkg);     // ハッシュ内に格納されたクラス定義を復元
rcv = Class.assign(rcv,cls);                    // ハッシュをクラスに割当て（オブジェクト化）
rcv.view();                                     // Class.assign()以後、rcv はインスタンスとして振る舞う
```

10Record クラス

クラスメソッドのみ提供する抽象クラス。

クラス・メソッド	
assign(obj:object, rec:record=null):object	obj で示すオブジェクトまたはハッシュを rec で示すレコードにアサインし、obj への参照を返す。以後、obj に対して rec レコードのメソッドを使用できる。rec を指定しない場合は obj の持つ同名のレコードを指定したものとみなす。 正しくない rec を指定した場合は null を返す。
toHash(obj:object):object	obj オブジェクトおよび obj が参照するすべてのオブジェクトをハッシュ化し、obj への参照を返す。
toInstance(obj:object, rec:record=null):object	obj で示すハッシュを rec で示すレコードのインスタンスとして構築し、obj への参照を返す。 rec を指定しない場合は、obj が持つ同名のレコードを指定したものとみなす。 rec が存在しない場合は obj 内容は変更されない。

10.1Record クラスメソッドの使用例

Record.toHash()と Record.toInstance()は DBMAN を使用して record インスタンスを格納したり、また読み出したインスタンスを復元する場合に使用する。
またハッシュ化したデータを他のコンピュータに送信し、受信側でこれをインスタンスに復元して取り扱う場合も効果的である。
例：

```
record  foo{
        field   value:string;
        method  view()  writeln(value);
}
var     dbman:DBMAN = new DBMAN("DataBase.nlx",DBMAN.CREATE);          // データベースを作成
var     rec:DBREC = dbman.newRecord("data");                           // “data”というレコードを作成
var     data = new foo{value = "データです";}                          // foo インスタンスを生成
rec.put(Record.toHash(data));                                          // foo インスタンスをハッシュ化して格納
dbman.close();                                                         // データベースを閉じる

dbman = new DBMAN("DataBase.nlx",DBMAN.READ);                          // データベースを開く
data = dbman.record("data").get();                                     // “data”レコードの内容を取得（foo というハッシュ）
data = Record.assign(data,foo);                                        // ハッシュを foo インスタンス化
data.view();                                                           // foo インスタンスの view()メソッドを利用
```

11Object クラス

Object クラスはユーザ定義クラスの暗黙基底クラスである。

※ クラスメソッドの引数には組み込みクラスならびにその派生クラスから生成したオブジェクトを指定してはならない。

クラス・メソッド	
assign(obj:object, cls:class):object	obj で示すオブジェクトまたはハッシュを cls で示すクラスにアサインし、obj への参照を返す。以後、obj に対して cls クラスのメソッドを使用できる。
asHash(obj:object):object	obj を以後パック化もしくはシリアライズする際にハッシュとして扱う。obj を返す。
asInstance(obj:object):object	obj を以後パック化もしくはシリアライズする際にインスタンスとして扱う。obj を返す。
duplicate(obj:object):object clone(obj:object):object	obj オブジェクトを複製した結果のオブジェクトを返す
getClass(obj:object):class	obj のクラスを返す。obj がインスタンスではない場合は null を返す。
getIdent(obj:object):int	obj オブジェクト固有の識別番号（整数値）を返す。
getMember(obj:object):string[]	obj オブジェクト内のメンバー名文字列リストを１次元配列として返す。
getName(obj:object):string	obj オブジェクトの識別名を文字列で返す。インスタンスならクラス名に同じ。

・duplicate()および clone()メソッドは、そのオブジェクト内のフィールドがオブジェクト参照である場合、参照するオブジェクトを複製せずに参照値のみをコピーする。（shallow copy）
・getMember()によって返されるメンバー名にはフィールドおよびメソッド、さらに基本クラスインスタンス名が含まれる。

12Array クラス

Array クラスは配列とハッシュに共通するオブジェクトを規定するクラス。

コンストラクタ	
Array()	空の配列オブジェクトを生成し、その参照値を返す。
Array(ary:Array)	ary で指定した配列オブジェクトを複製し、その参照値を返す。 例：new Array({1,2,3})は Array.duplicate({1,2,3})と同じ結果となる。
Array(arg:variant,,,)	第１引数が配列以外もしくは複数の引数が指定された場合、各々を要素とする配列を構築する。

クラス・メソッド	
duplicate(ary:Array):Array clone(ary:Array):Array	ary に指定した配列もしくはハッシュを、複製してその参照値を返す。 要素がオブジェクト参照である場合は参照値をコピーする。（shallow copy）
isArray(val:variant):bool	val が配列の場合のみ true を返す。
length(obj:object):int	obj に指定したオブジェクト内の配列要素数を返す。obj が配列でない場合は 0 を返す。
purge(ary:Array, normalize:bool=false):Array	ary に示す配列中の NULL 要素をすべて取り除き、以降の要素を前詰めする。 normalize に true を指定した場合は、重複する（同じ値の）要素も取り除く。

インスタンス・メソッド	
append(val:variant):variant	val を配列の最後の要素として追加し、val を返す。
assign(cls:class):object	cls で示すクラスのインスタンスフィールドに配列要素を順に埋め込んだオブジェクトを返す。
duplicate():Array clone():Array	自身を複製した結果の配列を返す。要素がオブジェクト参照である場合、参照するオブジェクトを複製せずに参照値のみをコピーする。（shallow copy）
getName():string	オブジェクトの名前を返す。
getMember():string[]	ハッシュメンバー名リストを１次元配列として返す。
getValue():Array	配列自身への参照を返す。
getValue(element:int):variant	要素番号 element で指定する配列要素を返す。 element に負値を指定した場合は、length 値を加算した結果の要素を返す。
join(term:string=””):string	配列の各々の要素を term で指定した区切り文字を付加して結合した結果の文字列を返す。 配列要素が文字列でない場合は、各々の要素に toString()メソッドを適用して得た文字列を結合する。
indexOf(value:variant=null, start:int=0):int	要素番号 start（負値の場合は length 値を加算した結果）以後の要素について、value で示す検索値と一致するもっとも小さい要素番号を返す。該当する要素がない場合は-1 を返す。 value に NULL を指定した場合は、空でない要素の要素番号を返す。

indexOf(check:method, start:int=0):int	要素番号 start（負値の場合は length 値を加算した結果）以降の要素から順に check で示すメソッドの引数に要素値を渡して呼出し、その結果が true であれば、その要素番号を返す。最終要素を引き渡してもなお false が返された場合は-1 を返す。
lastIndexOf(value:variant=null, end:int=0):int	要素番号 end（0もしくは負値の場合は length 値を加算した結果）直前の要素について、value で示す検索値と一致するもっとも大きい要素番号を返す。該当する要素がない場合は-1 を返す。 value に NULL を指定した場合は、空でない要素の要素番号を返す。
lastIndexOf(check:method, end:int=0):int	要素番号 end（0もしくは負値の場合は length 値を加算した結果）直前の要素から逆順に check で示すメソッドの引数に渡して呼出し、その結果が true であれば、その要素番号を返す。先頭要素を引き渡してもなお false が返された場合は-1 を返す。
length():int length:int	配列の長さ（最後の有効要素の番号＋１の値）を返す。 例：{3,2,,,10}.length() は 5 である。
length(limit:int):int	配列の長さを limit で指定する長さに縮め、その結果の長さを返す。 limit が配列の実際の長さより大きい場合は機能せず、小さい場合は最終要素が limit 以下になるまで要素を削除する。配列内に null 要素が含まれる場合、結果は limit よりも小さい場合もありうる。
matchOf(key:string, start:int=0):int	要素番号 start 以降の要素で、文字列 key を含むもっとも小さい要素番号を返す。 該当する要素がない場合は-1 を返す。 ※要素が文字列でないものは検索対象としない。
lastMatchOf(key:string, end:int=0):int	先頭から要素番号 end 直前までの要素で、文字列 key を含むもっとも大きい要素番号を返す。 該当する要素がない場合は-1 を返す。 end に 0 もしくは負値を指定した場合、length()値を加算した要素番号直前から検索を開始する。 ※要素が文字列でない場合は検索対象としない。
pop(start:int=0,length:int=0, remain:bool=false):Array	要素番号 start 位置から length 要素分の部分配列を取り出す。start に負値を指定した場合は配列の長さを加算した値を要素位置とする。length に 1 未満の値を指定した場合は start 位置以降のすべての要素を取り出す。remain に false を指定した場合は取り出した要素は元の配列から削除して前詰し、true を指定した場合は要素を削除しない。 ※ECMA-262 仕様の pop()と同様な動作をおこなうには pop(-1)[0]とすること。
push(ary:Array,element:int=0) :int	ary で指定する配列の要素を element 位置の要素直前に挿入し、挿入後の配列長さを返す。 ary 配列全体を１要素として挿入する場合は、ary を “ {} ” で括り、 “ {ary} ” とする。 element に負値を指定した場合は配列の長さを加算するが、結果がさらに負値となる場合は 0 として扱う。※ECMA-262 と仕様が異なる点に注意
push(val:variant,element:int=0) :int	配列以外の型である val を element 位置の要素直前に挿入し、挿入後の配列長さを返す。 element に負値を指定した場合は配列の長さを加算するが、結果がさらに負値となる場合は 0 として扱う。※ECMA-262 と仕様が異なる点に注意
purge(norm:bool=false):Array	値が null の要素を削除して前詰めし、さらに norm に true を指定した場合は全要素中に同じ値の

	要素を含まないよう削除して前詰し、配列自体の参照値を返す。 ※このメソッドは、配列を集合として扱う場合に使用すべきである
remove():variant	最後の要素を削除し、その要素値を返す。要素が空の場合は null を返す。
remove(element:int=-1):variant	要素番号 element の要素を削除して前詰めし、その要素値を返す。element に負値を指定した場合は length()値を加算する。 要素番号が配列の範囲を逸脱する場合は削除はおこなわず null を返す。
reverse():Array	要素を逆順に並び替え、その配列への参照を返す。
setValue(element:int=0, value:variant):variant	要素番号 element の要素に value 値を格納し、value を返す。 element に負値を指定した場合は配列の長さを加算した結果の要素を対象にするが、その結果でも負となる場合は機能しない。
shift():variant	先頭要素を取り除いて前詰し、その要素値を返す。結果として配列の長さは-1 される。
slice(start:int=0,end:int=null) :Array	要素番号 start から end 直前までの要素を取り出した配列を返す。配列には影響を与えない。 start を指定しない場合は 0、end を指定しない場合は配列の長さを採用する。各々に負の値を指定した場合は配列の長さを加算した結果を採用する。
splice(start:int=0,end:int=null, el1:variant=null,,,):Array	要素番号 start から end 直前までの要素を取り出した配列を返す。 取り出した要素は削除して前詰めし、第３引数以降に値を指定した場合はこれを start 以降に要素として挿入する。 start を指定しない場合は 0、end を指定しない場合は配列の長さを採用する。各々に負の値を指定した場合は配列の長さを加算した結果を採用する。
sort(comp:method):Array	配列要素各々につき、comp で示す比較関数の第１引数に低位要素を、第２引数に高位要素を格納して呼び出す。この関数が返す値が正数（整数もしくは実数）または論理値の真の場合に要素を入れ替える。※comp が数もしくは論理値以外を返した場合の動作は保証されない。
sort(rise:bool=true):Array	互いに比較可能な要素で構成する場合に限り、rise に true を指定した場合は昇順に、false を指定した場合は降順に配列をソートする。配列自身の参照値を返す。
subset(start:int=0, length:int=null) :Array	要素番号 start から length 要素を取り出した配列を返す。配列には影響を与えない。 start を指定しない場合は 0 とし、length を指定しない場合は start 以降のすべての要素を取り出す。start に負の値を指定した場合は配列の長さを加算した結果を start とする。
sumTotal():real	整数もしくは実数を値として持つ要素の合計を返す。
swap(m:int=0,n:int=0):Array	要素番号 m の要素と要素番号 n の要素を交換し、配列自身への参照を返す。 m,n とも負の値を指定した場合は配列の長さを加算した結果の要素番号とするが､結果的に負値となった場合は機能しない。
toString():string	配列もしくはハッシュ内容を文字列化して返す。配列には影響を与えない。
unshift(arg:variant,,,):Array	配列の先頭に、引数群に指定した値をその順に挿入し、その配列への参照を返す。配列の長さは追加した要素分増加する

13Wrapper クラス

Wrapper クラスは、プリミティブ（整数、実数、論理値、文字列）およびオブジェクトを内包値として保持するオブジェクトを生成する。
以下のクラスが用意されている。

クラスとコンストラクタ	コンストラクタ例	
Integer(arg:int) Integer(arg:Integer)	new Integer(100)	コンストラクタ引数は整数もしくは Integer インスタンス
Real(arg:real) Real(arg:Real)	new Real(0.5)	コンストラクタ引数は実数もしくは Real インスタンス
Number(arg:number) Number(arg:Number)	new Number(0.5)	コンストラクタ引数は整数、実数、Number インスタンス
Boolean(arg:bool) Boolean(arg:Boolean)	new Boolean(true)	コンストラクタ引数は論理値もしくは Boolean インスタンス
String(arg:string) String(arg:String)	new String("ABC")	詳細は String クラスを参照
StringBuffer(arg:string)	new StringBuffer("ABC")	詳細は StringBuffer クラスを参照
Exception(arg:variant) Exception(arg:Exception)	new Exception()	コンストラクタ引数には任意の値を指定できる（※1）
Variant(arg:variant) Variant(arg:Variant)	new Variant({1,2})	コンストラクタ引数には任意の値を指定できる（※1）

※1　引数に Wrapper クラスまたは Wrapper 派生クラスのインスタンスを指定すると、その内包値を渡した場合と同じとなる。

上記クラスのインスタンスに共通する以下の既定フィールドと既定メソッドがある。
setValue()メソッドの引数は各々のクラス・インスタンスもしくは対応するプリミティブ型に既定されている。

インスタンス・フィールド	説明
Value:variant	内包値

インスタンス・メソッド	
equalTo(obj:object):bool	obj と、型および内包値が等しい場合 true を返す
sameAs(val:variant):bool	val と、内包値が等しい場合 true を返す
getObject():object	このオブジェクトへの参照を返す
getValue():variant	オブジェクト内の内包値 Value を返す
setValue(val:variant):variant	引数 val に与えた値を、内包値として設定する。リターン値は val そのもの。
toString():string	"wrapper ＜クラス名＞{＜内包値を文字列化した値＞}"を返す

Integer クラスと Real クラス、Number クラスは以下のクラスメソッドを持つ。

クラス・メソッド	
Integer.parseInt(txt:string):int Number.parseInt(txt:string):int	txt の先頭文字から整数表記とみなせる範囲までを整数として評価してその値を返す
Real.parseReal(txt:string):real Number.parseReal(txt:string):real	txt の先頭文字から実数表記とみなせる範囲までを実数として評価してその値を返す
Integer.BIT(pos:int):int Number.BIT(pos:int):int	(1 << pos)の値を返す。pos が整数でないか 0..31 の範囲以外なら null を返す
Integer.format(value:int, radix:int=10, fig:int=0, term:string=","):string	value を radix 進数文字列化し、下位から fig 桁ごとに term 文字列で区切った文字列を返す。 radix は 2 以上 36 以下でない場合は 10 として扱い、10 かつ value が負の場合のみ“ - ”記号で先導する負値表現とする。fig が 1 以上 31 以下でない場合は区切りをおこなわない。 value は整数もしくは Integer インスタンス、あるいは値として整数をもつ Number インスタンスでなければならず、これら以外を指定した場合は null を返す

Exception クラスは以下のクラスメソッドを持つ。

クラス・メソッド	
Exception.at():string	catch ブロック内で実行することにより、例外を発行した位置のスクリプト・ソースファイル名と行番号を”:”で結合した文字列として得る。（例：”index.nal:10”） catch ブロック外で実行した場合は””を返す。 ※コンパイル済のソースファイルの場合、行番号はテキストソースとは必ずしも一致しない。

13.1Wrapper クラスの拡張例

プログラム中において値の意味付けを明確におこなうことを目的にする例を示す。

```
method  addAddress(name:string,addr:string){ ... }
```

の場合、メソッドの提供者は addAddress(“田中”,”東京”)という記述を期待しているはずであるが、利用者が間違えて addAddress(“東京”,”田中”)としても実行エラーとはならず、期待した結果にはならない。

これを

```
class Name extends String;
class Addr extends String;
method addAddress(name:Name,addr:Addr){ ... }
```

として提供し、addAddr(new Name(“田中”),new Addr(“東京”))と記述するように規定すれば引数の型付けは厳密におこなわれ、プログラムミスを容易に見つけやすくなる。

Wrapper クラスを拡張する場合、コンストラクタ引数の宣言も、また基本クラスコンストラクタの呼び出しも不要である。

14String クラス

文字列(string)を内包する Wrapper クラス

コンストラクタ	
String(str:string="")	str に指定する文字列をインスタンス内に保持する
String(strobj:String)	strobj が内部に保持する文字列をインスタンス内に複製して保持する。コピーコンストラクタ

クラス・メソッド	
parse(str:string,spec:string) :StringParse	str 文字列を spec 表記仕様に照らして解析した結果を StringParse オブジェクトとして返す。（※下記参照）

インスタンス・メソッド	
append(str:string=""):this	str をインスタンス内文字列の最後に追加する。
remove(len:int=1):string	最後から len 文字を切り取って返す。インスタンス内の値は切り取った残りとなる。 len がインスタンス内文字列長以上の場合は全体を抜き出し、インスタンス内は""となる。 len に 1 未満の値を指定した場合は 1 として扱う。
shift(len:int=1):string	先頭から len 文字を切り取って返す。インスタンスの値は切り取った残りとなる。 len がインスタンス内文字列長以上の場合は全体を抜き出し、インスタンス内は""となる。 len に 1 未満の値を指定した場合は 1 として扱う。
unshift(str:string=""):this	str をインスタンス内文字列の先頭に挿入する。
その他	Wrapper クラスおよび string プリミティブクラスを参照

14.1parse()メソッドの詳細

String.parse(source:string,spec:string):StringParse は引数として与えた文字列 source（以後ソースという）を、spec に指定する表記仕様に従って解析し、解析結果を下記に示す StringParse 型のオブジェクトとして返す。

```
object StringParse{
       field  parsed: string;      // spec に示される仕様に従ってソースを解析した結果の文字列
       field  remain: string;      // 解析されていない残りのソース文字列
       field  specify:string;      // 全ての解析が終われば""が、解析途中で不一致が検出されれば残りの表記仕様
       field  result: object{      // spec に記述された抽出キーセットに従った解析結果をフィールド要素として持つ
              ・・・
              }
       }
```

parsed には spec に示される仕様に従ってソースを解析した結果の文字列を、remain には解析されていない残りのソース文字列が格納さ

れる。
specify には全ての解析が終われば""が、解析途中で不一致が検出されれば残りの表記仕様が格納される。
result フィールドはハッシュオブジェクトであり、その要素としては spec に記述された文字列抽出キー、整数抽出キー、実数抽出キーに指定されたキーセットが解析の結果として格納される。なお、parsed の内容はソースと同じではなく解析結果としての文字列となる。
すべてのソースについて spec の仕様に合致したかどうかの判定は specify の値が""であることおよび remain の値が""であることで確認する。specify には未処理の仕様が格納されているのでどこまで解析したかがわかるし、remain には解析されたソースの残りが格納されるため不一致となった場所あるいは解析対象にならなかった残りのソースがわかる。

spec は下記の「要素」の任意の組み合わせとして記述する。なお、要素は複数の半角スペース文字で区切ってもよい。
下記説明中の「解析対象」とは、ソース中の文字列先頭から始まる文字カーソルを想定し、そのカーソル位置にある文字を指す。
但し空白文字要素を除き、解析対象が空白文字である場合は空白以外の文字位置までカーソルは自動的に先送りされる。

要素	解説
一致要素	一致すべき要素並びを、またこれらの要素並びを組として "¦" で区切りその全体を、"<" と " >" で括る。 解析対象位置以降について、組の要素並び順に要素を順次評価し、その値が""となったものがあればそこで組としての評価を中断する。但し省略要素の場合は値が""でも不一致とはならず後続の要素を評価する。 組全体の値は各々の要素の値を結合したものとなるが、不一致要素があった場合は組全体の値も""となる。 組の評価中に不一致となった場合、¦ で区切られた後続の組があれば一致する組が見つかるまで順次後続の組を解析する。この要素の値は一致した組の値となり、すべての組で一致しなければ値が""となる。 一致した組があれば、解析対象位置は一致した組の値分だけ進む。 例：『<ABC¦XYZ>』 は解析対象位置の表記が"ABC"もしくは"XYZ"に限り、その一致した文字列を値とする。 意味としては「"ABC"か"XYZ"であれば･･･」と解釈される。
省略要素	省略可能な要素並びを "[" と "]"で括る。この要素の値はソース中で省略されていなかった要素の値を結合したものとなる。よってすべて省略されていれば""に、一部省略なら省略されていない部分の値がこの要素の値となる。 解析対象位置はこの要素の値分だけ進む。該当しない場合でもこの要素は一致したものと解釈する。 例：『 [ABC] 』 は解析対象位置の表記が"ABC"、"AB"、"A"の場合のみこの要素の値となる。意味としては「"ABC"か"AB"か"A"であれば･･･」と解釈され、該当しない場合でも次の要素についての解析を継続する。
文字列抽出	%s または %S に続いて一致要素が記述されていれば一致要素の結果を、一致要素の記述がなければ解析対象位置のリテラル文字列（"_"以外の半角記号を含まない文字列）を値とする。続いて「:キー名」が指定されていればこの文字列値を StringParse.result 内のキー名で示すフィールドに格納する。 %S は続く一致要素の判定において半角英字の大文字と小文字を区別しない。 解析対象位置はこの要素の値分だけ進む。 例：『%s<ABC¦XYZ>:key』は解析対象位置の表記が"ABC"もしくは"XYZ"であればその一致した文字列を、一致しなければ""を key フィールドに格納する。 例:『%S<[Monday]¦[Sunday]>:key』は解析対象位置の表記が大文字小文字関係なく"Monday"あるいはその省略か"Sunday"あるいはその省略であればその一致した文字列を、該当しなければ""を key フィールドに格納する。

	注意：%s または %S に続く一致要素記述および":"さらにキー名の間にスペースを入れてはならない。
整数値抽出	%d は解析対象位置にある整数表記文字列を抽出する。但し%d に続いて「値域指定[5]」があればその条件を満たす必要がある。整数表記でないか、値域を満たさない場合の値は""となる。値が""でない場合のみ、続いて「:キー名」が指定されていれば値を整数化して StringParse.result 内のキー名で示すフィールドに格納する。 解析対象位置はこの要素の値分だけ進む。 例：『%d{1,12}:month』は、解析対象位置に 1 以上 12 以下の整数表現があればこれを数値化して month キー値に格納する。一致しない場合はハッシュフィールドを作成しない。 注意：%d に続く値域指定および":"さらにキー名の間にスペースを入れてはならない。
実数値抽出	%f は解析対象位置にある実数表記文字列を抽出する。但し%f に続いて「値域指定」があればその条件を満たす必要がある。実数表記でないか、値域を満たさない場合の値は""となる。値が""でない場合のみ、続いて「:キー名」が指定されていれば値を実数化して StringParse.result 内のキー名で示すフィールドに格納する。 解析対象位置はこの要素の値分だけ進む。 例：『%g{0,0.9999}:value』は、解析対象位置に 0 以上 0.9999 以下の数表現があればこれを数値化して value キー値に格納する。一致しない場合はハッシュフィールドを作成しない。 注意：%f に続く値域指定および":"さらにキー名の間にスペースを入れてはならない。 %f の代わりに %g でもよい。
空白文字	%b は１文字の半角あるいは全角スペースを意味する。解析対象位置がスペース文字ならその値をを、そうでなければ""をこの要素の値とする。一般的には解析対象位置に必ずスペースを必要とするような表記を検査する場合に使用する。 解析対象位置はこの要素の値分すなわちスペース１文字分進む。 意味としては「１文字の空白があれば･･･」と解釈される。
エスケープ文字	"%<"、"%["、"%'"、"%%"、"%?" は各々"%"に続く１文字と扱い、要素メタ文字とはみなさない。 解析対象位置が%に続く１文字に一致すればその文字を、一致しなければ""を値とする。 解析対象位置はこの要素の値分だけ進む。 例：『<10%%¦20%<>』は、「"10%"もしくは"20<"であれば･･･」と解釈される。
任意文字列	.. は、これに続く一致要素あるいは文字列が解析対象位置以降にあればその位置まで解析対象位置を進める。 解析位置以降で後続要素が一致すれば、解析対象位置から後続要素解析後のカーソル位置までを一致文字列とし、後続要素がソース文字列の最後まで一致しなければ不一致となる。 .. に続く要素指定がない場合は、解析対象位置からソース文字列の最後まで一致する。 例：『start..end』は解析位置が"start"で始まれば"end"まで一致するが"end"がなければ一致しない。 注意：『start..[end]』のように".."に続けて省略要素を指定した場合は例え解析対象位置に省略要素の記述がない場合でも、すべて省略されているものと解釈し解析位置を進めない。
任意文字	? は、解析対象位置以降のスペース以外の任意の１文字と一致する。 対象文字をその値とし、解析対象位置は１文字分進む。

[5]値域指定は、{下限値,上限値}、{下限値}、{,上限値} の何れかの表現でなければならない。

	例：『AB?XY』は"AB-XY"でも"ABXXY"でもよいが"ABXY"は"Y"が不一致となる。 参考：『AB[?]XY』とすれば、加えて"ABXY"でも一致となる。
文字候補	解析対象位置以降のスペース直後にあるべき候補文字を"'"と"'"で括る。 "'"直後に"!"をつけると解析対象位置以降のスペース直後にあってはならない候補文字という意味となる。 候補文字は、候補となる文字を並べるか半角文字については'A..Z'のように範囲を指定することもできる。 文字"'"を候補とする場合は"%'"と記述し、文字"!"を候補とするには"%!"とするか先頭以外に記述する。 候補文字に半角スペースを記述しても候補としない。 対象文字が候補に含まれていれば（!指定では含まれていなければ）その文字を値とし不一致なら""を値とする。解析対象位置はこの要素の値分進む。 例：『'A..Z 0..9 %' '』は「A から Z の間か 0 から 9 の間か'なら…」と解釈される。 例：『'!A..Z 0..9 %' '』は「A から Z の間以外かつ 0 から 9 の間以外かつ'以外なら…」と解釈される。
文字・文字列	要素メタ文字『<['%?』に該当しない文字は、解析対象位置以降のスペース以外の文字と比較して一致すればその文字を、不一致なら""を値とする。一般には文字列で指定し、文字列としての一致を解析する。 解析対象位置はこの要素の値分だけ進む。 例：『氏名：』は「"氏名："なら･･･」と解釈される。

仕様記述例：

```
"%s<昭和¦平成>:元号 %d{1,99}:YY [<年¦/>] [%d{1,12}:MM [月]] 生[ま]れ"
```

『%s<昭和¦平成>:元号』は、"昭和"もしくは"平成"という記述で始まっていればこれを元号というキーに格納する、という意味になる。
『%d{1,99}:YY』は、1 以上 99 以下の整数であればこれを、YY というキーに格納する。
『[年¦/]』 は"年"か"/"、あるいは省略されていてもよいことを示す。
『[%d{1,12}:MM [月]]』は、"nn 月"または"nn"であるか、記述全体が省略されていてもよいことを意味する。（nn は 1 以上 12 以下）
『生[ま]れ』は、"生まれ"か"生れ"に合致すればよいことを意味する。

プログラミング例：

```
var spec:string = "%s< 昭和 ¦ 平成 >:元号 %d{1,99}:YY [ <年 ¦ / > ] [%d{1,12}:MM <月 ¦ / >] 生[ま]れ";
writeln(String.parse("平成  12 年 の出来事",spec));
// ↑ object StringParse{parsed:"平成 12 年",remain:"の出来事",specify:"生[ま]れ",result:object{元号:"平成",YY:12}} と表示
writeln(String.parse("昭和 27 年  2 月  生まれ",spec));
// ↑ object StringParse{parsed:"昭和 27 年 2 月生まれ",remain:"",specify:"",result:object{元号:"昭和",YY:27,MM:2}} と表示
writeln(String.parse("昭和 30 年 生れです",spec));
// ↑ object StringParse{parsed:"昭和 30 年生れ",remain:"です",specify:"",result:object{元号:"昭和",YY:30}} と表示
writeln(String.parse("平成 10 年 13 月 生れ",spec));
// ↑ object StringParse{parsed:"平成 10 年",remain:"13 月生れ",specify:"生[ま]れ",result:object{元号:"平成",YY:10}} と表示
writeln(String.parse("大正 6 年 6 月 生れです",spec));
```

```
// ↑ object StringParse{parsed:"",remain:"大正 6 年 6 月生れです",specify:" %d{1,99}:YY [ <年 ¦ / > ] [%d{1,12}:MM <月 ¦ / >] 生[ま]れ",result:object{元号:""}} と表示
```

15StringBuffer クラス

文字列および文字を扱う Wrapper クラス
インスタンス内で管理される＜文字カーソル＞が指す文字列中の文字について、以下のメソッドの対象とする。
＜文字カーソル＞は、その指す文字の＜文字位置＞を値として持ち、＜文字列始端＞と＜文字列終端＞を越えない。
＜文字列始端＞は先頭文字の直前を、＜文字列終端＞は最終文字の直後を意味する。
＜文字位置＞とは、文字列始端なら-1、先頭文字なら 0、文字列終端なら文字列を構成する＜文字数＞と同じ値である。
＜文字数＞は文字列を構成する文字の個数であり、文字の構成バイト数に関わらず１表記を 1 として数える。

コンストラクタ	
StringBuffer(str:string="")	str に指定する文字列をインスタンス内に保持する インスタンス生成直後の文字カーソルは文字列始端にある。すなわち cursor()値は-1

インスタンス・メソッド	
position(offset:int=0):this	文字カーソルを offset 位置に設定する。
seekto(diff:int=1):this	文字カーソルを diff 文字分移動する。
length():int	文字列の文字数を返す。（表記１文字を１と数える値であり、"ABC"は 3、"漢字"は 2）
cursor():int	文字カーソルの指す文字位置値を返す。
getchar():string	文字カーソル位置の１文字を返す。カーソルが文字列始端あるいは文字列終端にある場合は""
getstring(pos:int=0):string	文字カーソルに対し、pos が文字列始端側なら pos 位置から文字カーソル直前までの文字列を、pos が文字列終端側なら文字カーソル位置から pos 直前までの文字列を返す。 pos が負なら length()+1 を加算し、さらに負なら 0 とする。 pos が文字カーソルと同じ場合は""を返す。
top():string	文字カーソルを先頭文字（文字列始端の次）に移動しその文字を返す。文字列が空なら""を返す。 position(0).getchar()と同じ。
rewind():int	文字カーソルを文字列始端に移動し length()値を返す。position(-1).length()と同じ。
next(offset:int=1):string	offset 文字分文字カーソルを先送りし、その文字を返す。seekto(offset).getchar()と同じ。
back(offset:int=1):string	next()の逆の機能、すなわち next(-offset)と同じ。
ischar(charset:string):bool	文字カーソルが文字列始端にある場合にはカーソルを先頭文字に移動したうえで カーソル位置の文字が charset 内のいずれかの文字であればカーソルを１文字分進め true を返す。該当しない場合は false を返す。 charset には ASCII 文字に限り"A..Z"のように値域を指定することもできる。
isstring(str:string):bool	文字カーソルが文字列始端にある場合にはカーソルを先頭文字に移動したうえで カーソル位置の文字列パターンが str であればカーソルをそのパターンの次の文字位置に移動し true を返す。一致しなければ false を返す。

findchar(charset:string):string	文字カーソル以降に charset に内のいずれかの文字があればその位置に文字カーソルを移動し、ない場合は文字列終端に移動する。移動後の文字カーソル位置の文字を返す。 charset には ASCII 文字に限り"A..Z"のように値域を指定することもできる。
findstring(str:string, skip:bool=false):string	文字カーソル以降に str と同じ文字列パターンがある場合、skip が false ならカーソルをそのパターンの先頭に、true ならパターンの次に移動して str を返す。 パターンがみつからなければ文字列終端にカーソルを移動し、""を返す。
findstring(strs[]:string, skip:bool=false):string	文字カーソル以降に strs 文字列配列のどれかの要素と一致する文字列パターンがある場合、最も近くで一致するパターンに対し、skip が false ならそのパターンの先頭に、true ならパターンの次にカーソルを移動し、一致した文字列を返す。 一致するパターンがなければ文字列終端にカーソルを移動し、""を返す。
insert(str:string):int	文字カーソルが文字列始端にある場合にはカーソルを先頭文字に移動したうえで カーソル位置に str 文字列を挿入し、挿入した文字列の次にカーソルを移動してそのカーソル値を返す。
cut(len:int=1):string	文字カーソルが文字列始端にある場合にはカーソルを先頭文字に移動したうえで カーソル位置から len 文字（表記を１文字と数える）を切り取って前詰めし、切り取った文字列を返す。
setValue(avoid:string):void asPrimitive():void asProxy():void	何も機能しない。（Wrapper クラスのメソッドをオーバライド） ※StringBuffer 内の文字列はこのクラス固有メソッド（上記）によって管理されており、他の手段（例えば Wrapper クラスメソッドや string プリミティブメソッドなど）ではデータを操作できないようにするための措置。
その他	Wrapper クラスおよび string プリミティブクラスを参照

16Buffer クラス

バイトを単位とするメモリーベースのデータを扱うためのインターフェースを提供するクラス。

コンストラクタ	
Buffer(size:int)	size で指定するバイト長のメモリー領域をもった Buffer オブジェクトを生成する。 size を指定しないか 0 を指定した場合は既定の大きさの領域を生成する。 生成直後の有効データ長は 0 である。
Buffer(base64:string)	base64 で指定する Bese64 エンコードされた文字列を基に Buffer オブジェクトを生成する。
Buffer(buf:Buffer)	buf オブジェクトを複製する。（コピーコンストラクタ）

クラス・メソッド	
encode64(buf:Buffer):string	buf で示す Buffer オブジェクトの内容を BASE64 でエンコードした結果の文字列を返す
decode64(str:string):Buffer	str で示す文字列を BASE64 デコードし、展開した結果の Buffer オブジェクトを返す
loadSBS(sbs:string[,,]):Buffer	引数に指定する SplitBitStream 値を合成した Buffer オブジェクトを返す
toEUC(str:string):Buffer	str で示すシフトＪＩＳ文字列を EUC コード系に変換し、結果を Buffer オブジェクトで返す
toUTF8(str:string,bom:bool=true):Buffer	str で示すシフトＪＩＳ文字列を UTF8 コード系に変換し、結果を Buffer オブジェクトで返す。 bom に false を指定すると BOM を付加しない。
toUTF16(str:string, bigEndian:bool=false, bom:bool=true):Buffer	str で示すシフトＪＩＳ文字列を UTF16 コード系に変換し、結果を Buffer オブジェクトで返す bigEndian に true を指定した場合は Big Endian として変換し、指定が無いか false を指定した場合は Little Endian として変換する。 bom に false を指定した場合は BOM[6]は付加しない

インスタンス・メソッド	
toString()	Buffer オブジェクトの内部プロパティ状態値を文字列化して返す。
expand(intsiz:int=0):bool	intsiz に指定した整数値バイト分の領域を拡張し､連続領域として拡張できた場合に true を返す。
length():int	有効データ長を返す。
index():int	カレント index 値（先頭からのバイト位置）を返す。
index(newidx:int):int	newidx をカレント index 値として置き換え、置き換え前のカレント index 値を返す。
indexOf(ref:Buffer):int	カレント index 位置以降で、ref で示す Buffer オブジェクトのメモリーパターンと一致するメモリーパターンの、先頭からのバイト位置を返す。一致するパターンがない場合は-1 を返す。
indexOf(ref:string):int	カレント index 位置以降で､ref で示す文字列のメモリーパターンと一致するメモリーパターンの、先頭からのバイト位置を返す。一致するパターンがない場合は-1 を返す。

[6] BOM（Byte Order Mark）については Unicode に関する文献を参照のこと

lastIndexOf(ref:Buffer):int	カレント index 位置以前で、ref で示す Buffer オブジェクトのメモリーパターンと一致するメモリーパターンの、先頭からのバイト位置を返す。一致するパターンがない場合は-1 を返す。
lastIndexOf(ref:string):int	カレント index 位置以前で、ref で示す文字列のメモリーパターンと一致するメモリーパターンの、先頭からのバイト位置を返す。一致するパターンがない場合は-1 を返す。
search(refs[]:Buffer, top:int=index(), end:int=length()):int	top 位置以降 end 直前までに、refs で示す配列要素 Buffer オブジェクトのメモリーパターンと一致するパターンがある場合、最も top 寄りで一致する refs 配列**要素の番号**を返す。一致する要素がない場合は-1 を返す。
search(refs[]:string, top:int=index(), end:int=length()):int	top 位置以降 end 直前までに、refs で示す配列要素文字列のメモリーパターンと一致するパターンがある場合、最も top 寄りで一致する refs 配列**要素の番号**を返す。一致する要素がない場合は-1 を返す。
maxLength():int	連続するバッファメモリーのバイトサイズを返す。
getByte():int	1 バイト取り出し、符号なし整数値として返す。（0～255）
getUINT16():int	Intel 形式符号なし 16 ビット整数として取り出し正の整数値として返す。
getUINT32R():real	Intel 形式符号なし 32 ビット整数として取り出し正の**実数値**として返す。
getShort():int	Intel 形式符号付 16 ビット整数として取り出し整数値として返す。
getLong():int	Intel 形式符号付 32 ビット整数として取り出し整数値として返す。
getFloat():real	Intel 形式単精度実数（32 ビット）として取り出し実数値として返す。
getDouble():real	Intel 形式倍精度実数（64 ビット）として取り出し実数値として返す。
getChar():string	１文字取り出し文字列として返す。（マルチバイト文字対応）
getString():string	改行文字（"\n"）までか、バイナリ 0 のバイトデータ直前までを文字列として返す。
getString(idx:int=index(), limit:int=length()-index(), noline:bool=false) :string	idx をカレント index に設定し、その位置から limit バイトを文字列として取り出す。 limit を指定しない場合はカレント index 以降の有効データ全体を対象とする。 対象となるデータ中に以下の条件が当てはまる場合は、limit バイト以下の文字列が返される。 noline に true を指定した場合はバイナリ 0 のバイトデータ直前までを文字列として取り出し、 false を指定した場合は改行文字（'¥n'）までか、バイナリ 0 のバイトデータ直前までを取り出す。 （改行文字は取り出した文字列の最後の文字として含まれる）
getHexPtn(idx:int=index(), limit:int=length()-index(), spc:bool=false):string	idx をカレント index に設定し、その位置から limit バイト範囲のバイト列を HEX 文字列として取り出す。limit を指定しない場合は idx 位置以降の有効データ全体を対象とする。 spc に true を指定した場合は 1 バイト分ごとに半角スペースを埋め込む。
getSBS(ptn:int):string	Buffer オブジェクト内容を ptn で示すビットパターンに対応する SplitBitStream 文字列として返す。ptn は下位 8 ビットのみ有効。
getText():string	Buffer オブジェクト内容を Shift-JIS 文字列とみなしてバイナリ 0 直前までを返す。
getEUCText():string	Buffer オブジェクト内容を EUC コードとみなし、これを Shift-JIS 文字列化した結果を返す。
getUTF8Text():string	Buffer オブジェクト内容を UTF8 コードとみなし、これを Shift-JIS 文字列化した結果を返す。

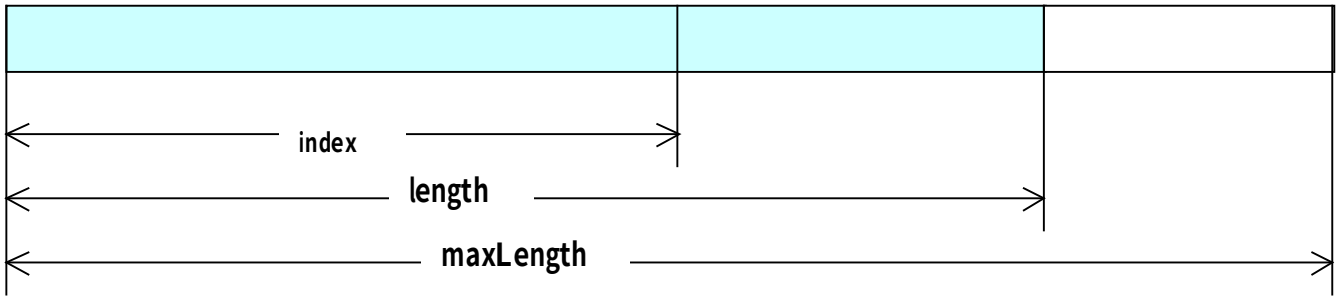
getUTF16Text(asBE:bool=false):string	Buffer オブジェクト内容を UTF16 コードとみなし、これを Shift-JIS 文字列化した結果を返す。 Buffer オブジェクトの先頭に BOM が無い場合のみ asBE が意味を持ち、BigEndian として扱う(true)か LittleEndian として扱う(false)かを指定する。
putByte(intVal:int):this	intVal の下位 1 バイトを格納する。
putShort(intVal:int):this	intVal の下位 16 ビットを Intel 形式で格納する。
putLong(intVal:int):this	intVal の 32 ビットを Intel 形式で格納する。
putFloat(realVal:real):this	realVal を単精度実数（32 ビット）として Intel 形式で格納する。
putDouble(realVal:real):this	realVal を倍精度実数（64 ビット）として Intel 形式で格納する。
putString(strVal:string):this	strVal で指定する文字列を格納する。
putHexPtn(hexStr:string):this	hexStr で指定する HEX パターンで表現されるバイナリデータブロックを格納する。 HEX パターンは 16 進表記（0‥9,A‥F,a‥f：HEX 文字）2 文字で 1 バイトを表すか、1 文字＋HEX 文字以外で 1 バイトを表現する。HEX 文字以外の文字は無視する。 例："12"⇒0x12　"b,"⇒0x0B　"3142,,33"は、"1B3"を putString()した場合と同じ結果
fill(byteValue:int=0, len:int=length()-index()):this	len バイト分を byteValue の下位１バイトのデータで埋める。 len を指定しない場合はカレント index 位置以降の有効データ全体を対象とする。 カレント index 位置は len バイト進む。
insert(buf:Buffer):this	buf に指定する Buffer オブジェクト内の全有効データをカレント index 位置に挿入し、カレント index 位置は buf.length()分進む。
insert(str:string):this	str に指定する文字列データをカレント index 位置に挿入し､カレント index 位置は挿入したバイトサイズ分進む。
replace(buf:Buffer):this	カレント index 位置以降のデータを、buf に指定する Buffer オブジェクト内の全有効データで置換する。カレント index 位置は buf.length()分進む。
delete(idx:int=index(), len:int=length()-index()):this	idx をカレント index とし、その位置から len バイトを削除し前詰めする。 len を指定しない場合は idx 以降の有効データ全体を削除する。
subset(idx:int,len:int):Buffer	idx 位置から len バイト分の内容を複写生成した Buffer オブジェクトを返す。len を指定しない場合は idx 以降の全有効データを取り出す。 インスタンスに影響は与えない。
BIT(pos:int,set:bool=null):bool	先頭バイトの LSB を位置 0 とする pos 位置のビット状態を、1 なら true、0 なら false で返す。 set に true を指定すると test&set し false では test&reset、指定しないと test のみ。
CRC16():int	有効データ範囲を対象として CRC16 演算した結果を返す。
CRC32():int	有効データ範囲を対象として CRC32 演算した結果を返す。
MD5():string	有効データ範囲を対象として MD5 メッセージダイジェスト化した 32 バイトの HEX 文字列を返す。

※ 上記水色背景の get 系および put 系メソッドにおいて
get 系メソッドの引数に整数値を指定すると、カレント index を指定された値に変更後 get をおこなう。

put 系メソッドの第２引数に整数値を指定すると、カレント index を指定された値に変更後 put をおこなう。
get 系および put 系共に、index 指定がない場合はカレント index 位置を対象として get および put する。
実行後カレント index 位置は扱ったデータバイトサイズ分移動する。
※ get 系メソッドにおいてはカレント index 位置が有効データ範囲を逸脱している場合、RuntimeException 例外が発生し、カレント index は変化しない。

16.1Buffer オブジェクトの構成

Buffer オブジェクトは内部に、index プロパティ、length プロパティ、maxLength プロパティを持ち、maxLength 値（最大 2^{31}）バイトの連続したメモリー空間を保持する。



各値はそれぞれ、index()、length()、maxLength()メソッドのリターン値として取り出すことができる。
index 値は生成時には 0 であり、get 系および put 系メソッドの呼出しによって変更され、これをカレント index という。
get 系メソッドでは取り出したデータの次の位置に、put 系メソッドでは格納したデータの次の位置になる。
また index 値は、index()メソッド引数に正整数値を指定して値を変更することができる。
length 値は生成時には 0 であり、put 系メソッドおよび delete メソッドの呼出しによって変更される。
maxLength 値は生成時に設定され、expand メソッドによって新たな連続領域を再構成して更新される。また、put 系メソッドによって length が maxLength を超える事態になれば自動的に拡張される。
length 値までの範囲のメモリー領域が有効データであり、get 系メソッドおよび delete、subset、indexOf メソッドの対象となる。
get 系メソッド呼出しにおいて有効範囲を逸脱するデータ長を指定した場合は、文字列 get 系メソッドおよび subset()メソッドにおいては対象とするデータは有効データ範囲を越えないが、getByte()など数値 get 系メソッドにおいては try-catch 構文中では≪OutOfBoundsException≫例外を発生し、try-catch 構文外では null を返す。

16.2Buffer オブジェクトの操作

Buffer オブジェクトへのデータ格納は put 系メソッドを使用し、副作用として index 値と、場合によっては length 値が更新され、さらに length 値が maxLength を越える場合は自動的に領域が拡張される。（※新領域への全コピーと旧領域の破棄がおこなわれる）
index 値は get 系メソッドおよび put 系メソッドの両方でアクセス対象となるデータの先頭位置を示し、各々のメソッド完了後は対象とし

たデータ長だけ後方移動（値の大きくなる方向）する。

Buffer オブジェクトは expand()メソッドの引数に指定する正整数バイト長分連続領域を拡張する。（但し全体で 2^{31} バイトを超えない）
例：

```
var buf = new Buffer(1000);
writeln(buf.maxLength());                    // 結果は１０００
buf.expand(1000);
writeln(buf.maxLength());                    // 結果は２０００
```

16.3Buffer オブジェクトの複製

Buffer オブジェクトは duplicate()メソッドを使用して、そのリターン値として複製された Buffer オブジェクトを得る。
この場合、データおよび管理情報すべて同じ値を引き継ぐ。
例：

```
var buf1 = new Buffer(1000);                 // ①長さ１０００バイトのバッファを生成する
buf1.putString("This buffer size = 1024Byte");  // ②生成したバッファに文字列を格納する
var buf2 = buf1.duplicate();                 // ③バッファを複製する
buf2.index(0);                               // ④index を０にリセットする（※）
writeln(buf2.getString());                   // ⑤複製されたバッファから文字列を取り出す
buf1 = null; buf2 = null;                    // ⑥バッファを削除する
```

※複製されたオブジェクトは index 値もそのまま継承するため、index(0)を実行して先頭にリワインドする必要がある。

16.4Buffer オブジェクト内メモリーイメージの比較

ふたつの Buffer オブジェクトが保有するメモリーイメージの比較（大きさおよびパターンの同一性）をおこなうには
var　buf1:Buffer, buf2:Buffer;
に対し、buf1.length() == buf2.length() かつ buf1.indexOf(buf2) == 0 あるいは buf2.indexOf(buf1) == 0 が成り立てば同じ大きさかつ同一メモリーパターンである。

16.5SplitBitStream の扱い

Buffer オブジェクトを特定のビットパターンで抽出したデータを SplitBitStream という。
例：

```
var     buf:Buffer = new Buffer();
buf.putString("ABC123");
var     sbs1:string = buf.getSBS(0x0F);          // bit0,1,2,3 を抽出
var     sbs2:string = buf.getSBS(0xF0);          // bit4,5,6,7 を抽出
```

これら sbs1,sbs2 はもとのデータ buf のビットサブセットとなっている。
これを再びもとのデータに復元するには
```
var    org:Buffer = Buffer.loadSBS(sbs1,sbs2);
```
とする。この後、org.getText()によって"ABC123"を得ることができる。
すなわち、loadSBS()クラスメソッドの引数として渡される SplitBitStream を合成する。
※SplitBitStream は文字列である。

16.6Buffer クラスの拡張

以下は、ＪＰＥＧ形式の画像データから、画像サイズ等を取り出すメソッドを提供するクラスを構成する例を示す。
例：

```
class   JPEG extends Buffer{
        public  method  imageSize():SizeInfo{
                private method  Int16():int     return  getByte() << 8 ¦ getByte();

                index(0);                               // カレント index を先頭に設定
                while(index() < length()){
                        select( Int16() ){
                          case  0xFFD8: ;               // StartOfImage
                          case  0xFFD9: return;         // EndOfImage
                          case  0xFFC0,0xFFC1,0xFFC2:{  // SOFn
                                index(index()+3);
                                return  new object SizeInfo{            // Make SizeInfo object & return it
                                        field   height = Int16(),       // 画像高さ
                                                width = Int16(),        // 画像幅
                                                size = length();        // Byte サイズ
                                        };
                                }
                          case  0xFFDA: return;         // SOS (後はデータ)
                          default:      index(index()+Int16());         // to next MARKER
                          } // end of select
                        } // end of while
                throw   new Exception("ERROR");                         // 正しいヘッダがなければエラー例外を通知
                }
        }
```

使い方例：

```
var     buffer:Buffer = new Storage("sample.jpg").get();        // JPEG 画像ファイルをすべて読み込む
var     jpeg:JPEG = new JPEG(buffer);                           // JPEG インスタンスとして構築する
var     size:SizeInfo = jpeg.imageSize();                       // ImageSize()メソッドを呼び出す
writeln("高さ=%d / 幅=%d / サイズ=%d".import(size.height , size.width , size.size));
```

17Type クラス

定数フィールドのみを提供する抽象クラス。クラス拡張はできない。

クラス・コンスタント	値	説明
Type.int	“INT”	整数型
Type.real	“REAL”	実数型
Type.string	“STRING”	文字列型
Type.bool	“BOOL”	論理型
Type.array	“ARRAY”	配列
Type.object	“OBJECT”	オブジェクト
Type.class	“CLASS”	クラス
Type.record	“RECORD”	レコード
Type.module	“MODULE”	モジュール
Type.method	“METHOD”	メソッド
Type.function	“FUNCTION”	関数
Type.null	“NULL”	ＮＵＬＬ型

17.1Type クラスの使用例

Type クラスは、typeof()演算子の結果を識別する目的に使用する。
例：

```
var     ary = {1 ,“ABC”, true , 0.5};
foreach(var item in ary){
        select(typeof(item)){
        case    Type.int:       writeln(“整数”);
        case    Type.real:      writeln(“実数”);
        case    Type.string:    writeln(“文字列”);
        case    Type.bool:      writeln(“論理値”);
        }//end select
}
```

18Date クラス

日付時刻を扱うクラス
太陽暦（グレゴリオ暦）で西暦１５８２年１０月１５日から西暦１００００年１２月３１日の範囲のみ対応。

コンストラクタ	
Date()	システム現在時刻をもとに、現在地時刻系列 Date オブジェクトを得る
Date(y:int,m:int=0,d:int=0, h:int=0,mm:int=0,s:int=0)	現在地時刻系列として指定した値の Date オブジェクトを得る。 y は 1582 以上 10000 以下を指定し、西暦年を意味する。 m は 0 から 11 で指定し、0 が１月を意味する。 d は 1 から 31 で指定し、m で指定した月の最終日を越えない。 h は 0 から 23、mm、s は 0 から 59 を指定する 各値に規定値外を設定した場合は正規化される。
Date(msec:real)　（廃止予定）	UTC 系 1970/1/1 0:0:0 からの経過ミリ秒をあらわす実数値 msec を指定して現在地時刻系列 Date オブジェクトを得る。
Date(date:Date)	date に指定する Date オブジェクトを複製する。コピーコンストラクタ

クラス・メソッド	
splitPeriod(period:real):int[]	経過日数を意味する period 値を日、時、分、秒、ミリ秒に分解した整数配列を返す。
timeRatio(h:int,m:int=0,s:int=0) :real	0:0:0 からの経過時間を意味する h 時間 m 分 s 秒を 1 日の時間割合として返す。 すなわち (h*3600.0+m*60.0+s)/(24*3600)の計算結果

インスタンス・メソッド	（時刻系状態に沿った作用および結果を返すもの）
isUTC():bool	Date オブジェクトの時刻系が UTC 系なら true を、Local 時刻系なら false を返す
toUTC()、toGMT()	Date オブジェクトの時刻系を UTC 系に切り替える。
toLocal():bool	Date オブジェクトの時刻系を Local 時刻系に切り替える。
addYear(year:int):this	year を年に加算し、結果の日が月の最終日を越える場合は最終日にする（閏年調整）
addMonth(month:int):this	month を月に加算し、結果の日が月の最終日を越える場合は最終日にする（大小月調整）
addDate(date:int):this	date を日に加算後正規化する
addHours(hous:int):this	hours を時に加算後正規化する
addMinutes(mins:int):this	mins を分に加算後正規化する
addSeconds(sec:int):this	sec を秒に加算後正規化する
addPeriod(period:real):this	経過期間を加算し正規化する。（経過期間については getValue()参照） 例：period に 1.5 を指定した場合１日と１２時間後となる
getYear():int	西暦年を返す

getMonth():int	月インデックスを 0 から 11 で返し、0 が１月、11 が１２月を意味する
getDate():int	1 から 31 を返し、日付を表す
getDay():int	曜日インデックスを返す（0 は日曜を意味し、以下順に 6 が土曜を意味する）
getHours():int	0 から 23 で時を返す
getMinutes():int	0 から 59 で分を返す
getSeconds():int	0 から 60 で秒を返す（60 が返される場合は閏秒の場合のみ）
getMilliSeconds():int	0 から 999 でミリ秒を返す
getYMDHMS():int[]	年、月、日、時、分、秒に要素分解した整数値配列を返す
getElapseSeconds():int	午前０時０分０秒からの経過秒数を返す
getElapseDays():int	年初からの経過日数を返す
getElapseMonths():int	getYear()*12+getMonth()値を返す
getMonthDay():int	28 から 31 で、保有年月についての月の最大日（日数）を返す 例：new Date(2000,1).getMonthDay() => 29
getDays():int	暦基準日からの経過日数を返す。（getDay() == getDays()%7 である）
getMonthName():string	月を"Jan","Feb",,,,"Dec"の文字列で返す
getDayName():string	曜日を"Sun","Mon",,,"Sat"の文字列で返す
getValue():real	暦基準日からの経過期間を返す。経過期間値は、整数部が日数を、小数部が 0:0:0 からの経過ミリ秒を 24*3600*1000 で割った値を表す
setYear(year:int):this	西暦年 year 年に置換し、日が月の最終日を越える場合は最終日にする
setMonth(month:int):this	month 月に置換して年月を正規化し、日が月の最終日を越える場合は最終日に調整する
setDate(date:int):this	date 日に置換して正規化する
setHours(hous:int):this	hours 時に置換して正規化する
setMinutes(mins:int):this	mins 分に置換して正規化する
setSeconds(sec:int):this	sec 秒に置換して正規化する
setMilliSeconds(msec:int):this	msec の 1000 の剰余をミリ秒の値として置換する
setYMDHMS(y:int,m:int,d:int, h:int,mm:int,s:int):this	setYear()から setSeconds()をまとめて実行する。 各々該当する値を null とした場合は該当値を変更しないが、下位値の設定値が各々の値域を越えている場合は正規化によって変更される。mSec は 0 となる。 例：date.setYMDHMS(,,1,0,0,0);は date の年月は変更せずに１日 0:0:0 に設定する。 例：date.setYMDHMS(); はミリ秒のみ 0 にする
setElapseSeconds(secs:int):this	secs を 0:0:0 からの経過秒とみなして正規化する。mSec は 0 にリセットされる
setValue(rday:real):this	インスタンスを、getValue()の結果と同じ意味を持つ rday 値に設定する

toString():string	"*week month date hours*:*minutes*:*seconds UTCzone year*"書式の文字列を返す week は"Sun"から"Sat"、month は"Jan"から"Dec"、date は"1"から"31"、hours は"0"から"23"の 24 時制、minutes は"0"から"59"、seconds は"0"から"60"ただし 60 は閏秒のある場合のみ、UTCzone は UTC 系では"UTC" Local 時刻系では"UTC"に続けて+/-に先導される 2 桁の時オフセットと 2 桁の分オフセット（例：日本標準時では"UTC+0900"）、year は西暦年
toString(format:string):string	format 文字列中の'$'で始まる制御文字列を、これに従った内容に置換した結果を返す。（※結果の文字数が 256 バイトを越える場合の結果は保証されない） 制御文字列は、<$>[[-][0]<幅数>]<Y｜M｜D｜h｜m｜s｜t>で指定し、文字 'YMDhmst' がそれぞれ西暦年、月、日、時、分、秒、ミリ秒値に対応する。 例："日付時刻は$Y/$2M/$-2D $2h:$02m:$02s です" 幅数は 1 から 9 の範囲であり、値の桁数が幅数に満たない場合のみ、幅数の直前に'0'を付加すると値の左に'0'を埋め、'-'を付加すると値の右に半角スペースを埋める。何も付加しない場合は値の左に半角スペース文字を埋める。幅数を指定しない場合は値の桁数幅となる。 年は、'Y'に替えて'Z'を指定すると明治以降に限り元号(ex."平成 10")となる 月は、'M'に替えて'N'で英語略称(ex."Jan")、'O'で和暦(ex."睦月")となる 時は、'h'に替えて'i'で 12 時間制として扱い、0 時と 12 時は 12 時、13 時は 1 時となる 曜日は、'$W'で英語略称(ex."Fri")、'$X'で漢字略称(ex."金")となる 午前午後は、'$a'で"am"と"pm"、'$A'で"AM"と"PM"、'$B'で"午前"と"午後"となる 例：14 時 10 分の場合、"Bi 時$m 分"と指定すると"午後 2 時 10 分"となる ※和暦表示の明治 45 年 7 月 29 日以前も、陰暦ではなく太陽暦で表示する

インスタンス・メソッド	（時刻系状態に影響されないもの）
getTimezoneOffset():int	現在地時刻の UTC 系との時差と夏時間調整値を分単位で返す。 東経値が正である地域では負値であり、現在地時刻に加算した結果が UTC 時刻となる値である
getJulianDay():real	ユリウス積算日を返す ユリウス積算日とは整数部はユリウス暦基準日からの経過日数を、小数部はグリニッジ標準時 12:00 を１日の起点とした経過秒数を 24*60*60 で割った値を意味する
setJulianDay(jd:real):this	jd で示すユリウス積算日に該当する値に設定する。
getTime():real	UTC 1970/1/1 0:0:0 からの経過ミリ秒を返す
setTime(tim:real):this	UTC 1970/1/1 0:0:0 からの経過ミリ秒を意味する値 tim に対応する値に設定する
toGMTString():string	GMT(UTC)ベースで、"*week, month date year hours:minutes:seconds* UTC"書式文字列化して返す。week と month は英語略称、他の値は暦時刻表記
toUTCString():string	GMT(UTC)ベースで、"*YYYY-MM-DDThh:mm:ss*Z"書式の文字列を返す
toLocalString():string	現在地時刻ベースで、toString()と同じ書式に文字列化して返す

インスタンス・フィールド	（時刻系状態に沿った値）
Year:int	西暦年
Month:int	月の index 値(0..11)　規定値域を逸脱した値を代入した場合は正規化される
Date:int	日の値 (1..31)　年月値に対応した日付値を逸脱した値を代入した場合は正規化される
Hours:int	時 (0..23)　規定値域を逸脱した値を代入した場合は正規化される
Minutes:int	分 (0..59)　規定値域を逸脱した値を代入した場合は正規化される
Seconds:int	秒 (0..59)　規定値域を逸脱した値を代入した場合は正規化される
MilliSeconds:int	ミリ秒 (0..999)　規定値域を逸脱した値を代入した場合は正規化される
Value:real	getValue()のリターン値および setValue()の引数値と同じ

※注：コンストラクタによって生成された直後の Date オブジェクトは Local 時刻系としての値を保持している。
これを UTC 系に変換するには toGMT()もしくは toUTC()を適用し、再び Local 時刻系に変換するには toLocal()メソッドを適用する。
例：

```
var     now = new Date();
writeln(“現在地時刻=”,now.getHours(),”:”,now.getMinutes());
now.toGMT();
writeln(“GMT 時刻=”,now.getHours(),”:”,now.getMinutes());
now.toLocal();
```

上記は、現在地時刻と UTC 時刻を表示する。

18.1使用上の注意

Date オブジェクトは、内部的には暦基準日からの経過日数を３２ビット整数で、１日の午前０時０分からの経過秒数を３２ビット整数で保持している。そのため、setXXXX()メソッド、addXXXX()メソッドに指定する値は必ずしも正規化された値でなくてもよいが、内部的に正規化される過程で値域を逸脱することがないよう配慮する必要がある。
例：

```
var     day = new Date();
day.setHours(day.getHours()+1000000);     // 今から 1000000 時間後の日付時刻
```

上記は値域を越えるため正しい結果とならない。

カレンダーの適合範囲は西暦１５８２年１０月１５日から西暦１００００年１２月３１日までであり、これを逸脱する場合は月日と曜日は正しくない。（※参考：１５８２年１０月１５日はグレゴリオ暦への改暦日である）

18.2インスタンスの比較

Date インスタンスを比較する場合、その比較する意図に応じて採用するメソッドを適切に選択しなければならない。

1. **時刻系列としてのミリ秒単位の比較をおこなうには** `getValue()` **あるいは** `getTime()` **同士を比較する。**
2. **暦日を比較するには** `getDays()` **あるいは** `getJulianDay()` **同士を比較する。（もしくは** `getValue().toIntValue()`**同士）**
3. **通算年月を比較するには** `getElapseMonths()` **同士を比較する。**

19Math クラス

クラスメソッドおよび定数を提供する抽象クラス。クラス拡張はできない。

クラス・メソッド	
abs(x:number):number	絶対値を返す
acos(x:real):real	arccos()関数の結果を返す
asin(x:real):real	arcsin()関数の結果を返す
atan(x:real):real	arctan()関数の結果を返す
ceil(x:real):int	x に対し、数直線上のより大きい整数に切り上げた結果を返す（floor()参照）
comb(n:int,r:int):real	nCr（組合せ）を返す
cos(x:real):real	cos()関数の結果を返す
exp(x:real):real	exp()関数の結果を返す
fact(n:int):real	n!（n の階乗）を返す
floor(x:real):int	x に対し、数直線上のより小さい整数に切り下げた結果を返す（ceil()参照）
gcd(x:int,y:int):int	正整数 x と正整数 y の最大公約数を返す
log(x:real):real	log() 関数の結果を返す
max(x:number,y:number):number	x,y のうち値の大きいほうを返す
min(x:number,y:number):number	x,y のうち値の小さいほうを返す
odd(x:int):bool	整数 x が奇数の場合のみ true を返す
perm(n:int,r:int):real	nPr（順列）を返す
pow(x:number,y:number):real	x^yを返す
prime(x:int):int[]	正整数 x 以下のすべての素数を配列オブジェクトとして返す(3 ≦ X ≦ 10000000 であること)
random():real	呼び出しごとに 0 から 1.0 未満の異なる実数を返す
round(x:real):int	x の少数第１位を四捨五入した結果を返す
sin(x:real):real	sin() 関数の結果を返す
sqrt(x:real):real	x の平方根を返す
tan(x:real):real	tan() 関数の結果を返す
toChar(x:number):string	x の値に対応する文字コードで規定される１文字の文字列を返す

クラス・コンスタント	
E:real	自然対数 e の値（オイラー定数）
LN2:real	log(2)　（2 の自然対数）
LN10:real	log(10)　（10 の自然対数）
LOG2E:real	1/log(2)　（2 を底とする *e*（自然対数の底）の対数の値）

LOG10E:real	1/log(10) （10 を底とする *e* （自然対数の底）の対数の値）
PI:real	π の値
SQRT1_2:real	√0.5
SQRT2:real	√2

20Storage クラス

ローカルホスト内のファイルを扱うクラス。
サーバサイドＮＡＬおよびスタンドアローンＮＡＬのみ実装

コンストラクタ	
Storage()	インスタンスを生成する。
Storage(path:string, mode:int=Storage.READ)	インスタンスを生成し、open()メソッドを実行してファイルをインスタンスに連結する。 引数については open()メソッドと同じ。（※1）

クラス・メソッド	
enableExcept()	エラー時例外発生を許可する。既定値は disable 状態である。
disableExcept()	エラー時例外の発生を抑制する。
isExist(path:string):bool	path で指定するファイルが存在すれば true を返す
getSize(path:string):int	path で指定するファイルの大きさを返す。ファイルが存在しない場合は null を返す
getTime(path:string, KIND:string=null):real	path で指定するファイルの生成、更新、アクセス時刻を UTC 系 1970/1/1 0:0:0 からの経過ミリ秒をあらわす実数値で得る。この値を new Date()の引数に与えると Date オブジェクトを得る。 ファイルが存在しない場合は null を返す。 KIND に"ACCESS"を指定した場合はアクセス時刻、"CREATE"を指定した場合は生成時刻、その他の値もしくは指定しない場合は最終更新時刻を返す。
setTime(path:string, time:real=null):bool	path で指定するファイルの更新時刻を UTC 系 1970/1/1 0:0:0 からの経過ミリ秒をあらわす実数値 time に設定する。time を設定しない場合は現在時刻に設定する。 path に該当するファイルがないか、読出し専用属性となっている場合は false を返す。 ※現在 WRITE モードで開いているファイルに対しては設定できない。
remove(path:string):bool	path で指定したファイルを削除する。 削除できたら true を返し、削除できない場合は false を返す。 ※現在開いているファイルを削除してはならない。 ※SERVER モデルにおいては、拡張子“.nal”を削除することはできない。
rename(path:string, newname:string):bool	path で指定するファイル名を newname で指定する名前に変更する。成功した場合は true を返す。 path と newname のディレクトリ部が異なる場合はファイルの移動をおこなう。ただし存在しないディレクトリを指定してはならない。 ※SERVER モデルにおいては、拡張子“.nal”を対象にすることはできない。
mkdir(path:string):bool	path で指定するディレクトリを作成する。作成できない場合は false を返す。 path の最後は文字"/"で終わっていなければならない。 例："d1/d2/fl"を指定した場合に作成されるディレクトリは d1/d2 である。

rmdir(path:string):bool	path で指定する空のディレクトリを削除する。削除できない場合は false を返す。 path の最後に文字"/"があってもなくてもよい。
list(pathspec:string="*.*", attr:bool=false):string ※Windows 版のみ実装	pathspec で指定するパスに該当するファイル名を"/"で連結した文字列を返す。 attr に true を指定した場合はファイル名に続いて":"と、"A"=archive、"C"=compressed、 "D"=directory、"H"=hidden、"R"=readOnly、"S"=system を意味する文字列で連結する。 複数属性値を持つ場合は文字を結合する（例："AR"） 使用例： var list[]:string = Storage.list("C:/*.txt").split("/");

インスタンス・メソッド	
open(path:string, mode:int=Storage.READ):bool	ファイルを開き、ストレージと連結する。 以後、ストレージへの操作はファイルへの操作として機能する。 mode には Storage.READ、Storege.WRITE、Storage.CREATE を指定でき、目的に応じて各々をビットＯＲ（"|"）演算子で結合して指定する。 Storeag.READ のみ指定した場合は読み出し専用となる。 Storege.WRITE を指定した場合は読み書きを許可する。 Storage.CREATE を指定した場合は読み書きモードとなり、ファイルが存在しなければ新たにファイルを生成する。 また、Storage.MKDIR と共に指定すると path 中のディレクトリが存在しない場合これを自動的に作成する。 現在すでにファイルをオープンしている場合は一旦これをクローズしてからオープンする。 オープンに成功すれば true を、失敗すれば false を返す。（※1）
close()	ストレージを閉じる。以後のストレージ操作は open()メソッド以外すべて無効となる。 Storage インスタンスのデストラクタにより、インスタンス消滅時に自動的に実行される。
length():int	ストレージに連結されたファイルの先頭から EOF 位置までのバイト長さを返す。 EOF を超えて put()をおこなった場合は自動的に伸長される。（※2）
index():int	カレント index 値を返す。 カレント index はストレージ先頭からのバイト位置を示し、open()直後の値は 0 である。
index(idx:int):int	idx を新たなカレント index とし、変更前の index 値を返す。
lock():bool	ストレージに連結されたファイルを Lock する。 Lock できた場合は true を返し、すでに他のプロセスが Lock している場合は false を返す。 Lock 範囲はファイル全体を対象とする。
unlock():bool	Lock を解除する。 Lock していなかった場合は false を返す。

get():Buffer	カレント index 以降 EOF までのデータを読み出し、Buffer オブジェクトとして返す。 カレント index は EOF 位置まで進む。 ファイルがオープンされていない場合は null を返す。（※2）
get(length:int):Buffer	カレント index 以降の length バイトのデータを Buffer オブジェクトとして返す。 length にカレント index 以降の有効データ長以上の値を指定した場合は EOF までを読み出す。 カレント index は読み出したバイトサイズ分進む。 ファイルがオープンされていない場合は null を返す。（※2）
get(length:int,idx:int):Buffer	idx をカレント index としてから get(length)をおこない、その結果を返す。（※2）
getText():string	カレント index 以降 EOF までのデータを文字列として返す。 データ中にバイナリ 0 が含まれる場合は、結果の文字列はその直前までとなる。 カレント index は EOF 位置まで進む。 ファイルがオープンされていない場合は null を返す。（※2）
getText(length:int):string	カレント index 以降の length バイトのデータを文字列として返す。 データ中にバイナリ 0 が含まれる場合は、結果の文字列はその直前までとなる。 カレント index は EOF を越えない範囲で length バイト分進む。 ファイルがオープンされていない場合は null を返す。（※2）
getText(length:int,idx:int) :string	idx をカレント index としてから getText(length)をおこない、その結果を返す。（※2）
getTextLine(idx:int=null):string	idx をカレント index として、その位置から改行文字（"¥n"）までの文字列を読み出して返す。 idx を指定しない場合は現在のカレント index 位置からを対象とする。 データ中に改行文字がない場合はバイナリ０あるいは EOF までを返す。 改行文字"¥n"は結果の文字列中に含まれる。（※2）
put(buffer:Buffer, idx:int=null):int	idx をカレント index とし、buffer オブジェクトの内容をカレント index 以降に書き込む （カレント index は書き込んだバイトサイズ分進む） idx を指定しない場合は現在のカレント index 値位置以降に書き込む。 書き込みに成功した場合はそのサイズを返し、失敗した場合および Buffer に有効データがなかった場合は０を返す。（※3）
put(str:string, idx:int=null):int	idx をカレント index とし、str で示す文字列をカレント index 以降に書き込む。 その他の機能および副作用は Buffer オブジェクトを put する場合と同じ。（※3）
flush()	ストレージに書き込んだ内容を直ちに物理メディアに反映する
isEof():bool	カレント index が EOF 位置以降にあれば true を、前にあれば false を返す。
isOpen():bool	ファイルをオープンしていれば true, さもなくば false を返す。

クラス・コンスタント	
Storage.READ:int	open()の mode 引数に指定し、読込モードを意味する。

Storage.WRITE:int	open()の mode 引数に指定し、読み書きモードを意味する。
Storage.CREATE:int	open()の mode 引数に指定し、読み書きモードかつ、ファイルが存在しない場合には生成する。
Storage.MKDIR:int	open()の mode 引数に Storage.CREATE とともに指定し、パス中のディレクトリが存在しない場合にこれを作成してからファイルを作成する。
Storage.LOCK:int	open()の mode 引数に指定し、ファイルロックを意味する。

プラットホーム固有のランダム・アクセス・ファイルを扱う組込みクラスとして提供される。
ストレージは、そのファイルシステムの構成がどのようなものであっても、一意的な概念で扱うように統一したものであり物理的な構成に依存しない。

20.1ストレージの概念

ストレージは論理的に $2^{31}-1$ バイトまでの連続するバイト列として扱われる。
各々のバイトデータは index という指標でその位置を特定される。
ストレージへの操作は、任意の index 値で示す位置にあるデータブロック（１バイト以上の長さをもったバイト列）を Buffer オブジェクトに取り出し、または Buffer オブジェクトの内容を書き込む。

ストレージ・オブジェクトは、コンストラクタ引数にファイル・パスおよびアクセス方法を指定して生成する。
データの取り出しは get()、データの書きこみは put()メソッドでおこなう。
ファイルを閉じる操作は close()メソッドでおこなう。
close()メソッド実行後は、新たに open()メソッドを行なわない限り、すべての get()、put()メソッドは無効となる。

get()、put()メソッドは前もって index()メソッドによってデータ位置を指定するが、先行する get()、put()メソッドによって index 値は自動更新されるため、連続するデータをアクセスする場合は index()メソッドの実行を省略できる。

データの書き込みは、ファイルの範囲を超えて指定できるが、この場合、自動的に拡張された部分のデータのうち、実際に書き込みをおこなっていない部分のデータについては、多くのプラットホームでは無意味なデータでしかない。

ファイルの get()、put()が成功したかどうかについては、各々のメソッドのリターン値で判断する必要がある。
get()メソッドの場合、指定した大きさのデータ未満のデータしか取り出せない場合、ファイルの I/O エラーか EOF（End of File）の可能性があり、isEof()メソッドで true が返される場合は EOF であることがわかる。

20.2ファイルの操作例

•読み出し

```
var file = new Storage();                    // ストレージオブジェクトを生成する
```

```
file.open( “ /home/data/file1.dat ” );          // ファイルを読み出し専用で開く
var buf = file.get(file.length());              // データをすべて取り出す
file.close();                                   // ファイルを閉じる
```

•書き込み

```
var file = new Storage( “ /home/db/member.data ” ,Storage.WRITE);
file.index(REC_SIZE*target_count);              // 読み出しレコード位置を指定
var buf = file.get(REC_SIZE);                   // レコードを読み出す
buf.ZeroFill(NAME_FIELD_SIZE);                  // フィールドをクリア
buf.putString( “ new member name ” ,0);         // フィールドを更新
file.index(REC_SIZE*target_count);              // 書き込みレコード位置を指定
file.put(buf);                                  // レコードを書き込む
file.close();                                   // ファイルを閉じる
```

•コピー

```
var buf,src,dst;
var src_path = "/home/db/member.data";
var dst_path = "/home/backup/member.data";
Storage.remove(dst_path);                       // コピー先を先に削除する

src = new Storage(src_path);                    // ソースファイルを開く
dst = new Storage(dst_path,Storage.CREATE);     // コピー先ファイルを生成する

/*
 * 65536 バイトのバッファを介してソースを読み出し、
 * そのサイズが 0 になるまで（EOF 検出後は 0 になる）内容をコピー先に書き込む
 */
for(;(buf = src.get(65536)).length();) dst.put(buf);

src.close(),dst.close();                        // ソースとコピー先を閉じる
```

20.3メソッド呼び出しに伴う例外の発生

Storage.enableExcept()実行後から Storage.disableExcept()実行以前に、コンストラクタ内でファイルオープンに失敗した、あ

るいはファイルを開いていない（isOpen()の結果が false）状態で isOpen()以外のインスタンスメソッドを実行した場合に、IOException エラー例外を発行する。
例外値に getValue()メソッドを適用することによって文字列によるエラーメッセージを得ることができる。
例：

```
var     file = new Storage("userfile.txt");
Storage.enableExcept();                         // 例外の発行を許可
try{
        writeln(file.length());                 // ファイルがオープンできていない場合、例外を通知する
        file.close();
}
catch(err:IOException){
        writeln(err.getValue());                // エラー内容を表示する
}
```

各々のメソッドは以下の条件で例外を発行する。
※1　ファイルを開けない場合
※2　ファイルが開かれていない場合（isOpen() で false が返される状態）
※3　正しい書き込みができなかった場合

21DBMAN クラス

ローカルホスト内のデータベースを扱うクラス。
サーバサイドＮＡＬおよびスタンドアローンＮＡＬのみ実装

DBMAN クラスは、DBMAN クラスのインスタンスであるデータベースオブジェクト（型名：DBMAN）と、データベースオブジェクト内のメソッドによって生成されるレコードオブジェクト（型名：DBREC）によって機能する。
データベースオブジェクトは、データベース全体にわたる操作を担当し、レコードオブジェクトは個々のデータレコードを操作する。
なお、DBREC オブジェクトはインスタンスではないため、その正当性は getName()メソッドを適用した結果が"DBREC"であることを利用すること。

凡例：
下表において、色分けされたメソッドは、このメソッドを適用可能なオブジェクトを示す。

	データベースオブジェクトのみ
	データベースオブジェクトおよびレコードオブジェクト
	レコードオブジェクトのみ

コンストラクタ	
DBMAN()	インスタンスを生成する。ただし、open()メソッドの実行を成功するまでデータベースへのアクセスはできない。
DBMAN(path:string, mode:int=DBMAN.READ, psw:string="")	インスタンス生成後 open()メソッドを実行する。path、mode、psw 引数の意味は open()メソッドと同じ。なお、open()に成功したかどうかは、isOpen()にて調べる必要がある。

クラス・メソッド	
enableExcept()	エラー時例外発生を許可する。既定値は disable である。
disableExcept()	エラー時例外の発生を抑制する。

インスタンス・メソッド	
dblength():int	現在オープンしているデータベースのファイルサイズを返す。オープンしていない場合は-1
isOpen():bool	現在データベース・ファイルをオープンしていれば true, さもなくば false を返す。

open(path:string, mode:int=DBMAN.READ, psw:string=""):bool	path で指定するデータベース・ファイルを開き、DBMAN オブジェクトと連結する。 mode には DBMAN.READ、DBMAN.WRITE、DBMAN.CREATE、DBMAN.LOCK を指定でき、目的に応じて各々をビットＯＲ（“|”）演算子で結合して指定する。 DBMAN.READ のみ指定した場合は読み出し専用となる。 DBMAN.WRITE を指定した場合は読み書きモードとなる。 DBMAN.CREATE を指定した場合は DBMAN.WRITE に加え、データベース・ファイルが存在しないときには新たにデータベース・ファイルを生成する。 DBMAN.WRITE または DBMAN.CREATE を指定した場合は DBMAN.LOCK は自動的に指定される。 現在すでにデータベースをオープンしている場合は一旦これをクローズしてからオープンする。 オープンに成功すれば true を返す。 既存の保護付きデータベースは psw の一致検査をおこなうが、非保護の場合は psw の値は意味を持たない。 新規データベースを作成する場合、psw に""以外のパスワードを指定した場合は保護付きとなる。
close()	データベース・ファイルを閉じ、以後の操作は open()メソッド以外すべて無効となる。 なお、DBMAN インスタンスのデストラクタによって自動的に実行される。
lock():bool	現在開いているデータベース・ファイルを Lock する。Lock されたデータベースは Lock を解除するか close するまで、別の DBMAN インスタンスにて Write モードで開くことはできない。 Lock できた場合は true を返し、すでに他の DBMAN インスタンスが Lock している場合は Lock できるまで待ち合わせるか、または false を返す。 Lock 範囲はデータベース全域を対象とする。
unlock():bool	Lock を解除する。Lock していなかった場合は false を返す。
length():int	データベースオブジェクトもしくはレコードオブジェクトの管理するレコードの総数を返す。 DBMAN インスタンス（データベースオブジェクト）に適用した場合は第１階層レコードの総数を返し、DBREC インスタンス（レコードオブジェクト）に適用した場合は該当 DBREC が管理するレコードの総数を返す。
matchOf(key:string, idx:int=0):int	内包する下位階層レコードから、key で示す文字列をインデックス情報として含むレコードのオフセットを整数として返す。該当するレコードがない場合は-1 を返す。 idx を指定しない場合は先頭レコード（レコードオフセット 0）から検索対象とし、正整数 idx を指定した場合は、指定したレコードオフセット以降を検索対象とする。 （getIndex()、setIndex()を参照のこと） ※返値ならびに idx に指定する値が整数で示すレコードオフセットである点に注意すること。

newRecord(key:string, append:bool=false):DBREC	key で指定するキーワードを持つレコードを生成し、そのレコードオブジェクトを返す。 データベースオブジェクトに対して適用する場合は、第１階層レコードを生成する。 レコードオブジェクトに対して適用する場合は、そのレコードの下位層レコードを生成する。 append が false の場合は先頭レコードとして挿入し、true の場合は最後に追加する。 ※key として指定できる文字列は 64 バイト未満かつ、""であってはならない。
record(key:string, create:bool=false, append:bool=false):DBREC	key に指定する文字列をキーワードとして持つレコードのレコードオブジェクトを返す。 該当するレコードが存在しない場合は null を返す。 ただし create に true を指定した場合、該当するレコードが存在しない場合はレコードを生成し、そのレコードオブジェクトを返す。append の指定および機能は newRecord()と同じ。 ※key として指定できる文字列は 64 バイト未満かつ、""であってはならない。
record(idx:int):DBREC	idx に指定した整数をレコードオフセット[7]とするレコードのレコードオブジェクトを返す。 該当するレコードがない場合は null を返す。
recordset():DBREC[]	データベースオブジェクトもしくはレコードオブジェクトが保有する全てのレコードを順に、レコードオブジェクトを要素とする配列を構成して返す。
recordset(low:string,high:string, abort:bool=false):DBREC[]	データベースオブジェクトもしくはレコードオブジェクトが保有する全てのレコードのうち、low に指定する下限キー値以上かつ high に指定する上限キー値以下のキー値をもつレコードを抽出し、各々のレコードオブジェクトを要素とする配列を構成して返す。 abort に true を指定した場合は、上限値を逸脱したキー値をもつレコードを検出した時点で抽出を完了する。（レコードが昇順キーで構成されている場合に有効）
recordset(callback:method) :DBREC[]	データベースオブジェクトもしくはレコードオブジェクトが保有する全てのレコードを順に、 callback:method (rec:DBREC, results[]:DBREC): bool 仕様のユーザ定義コールバックの第１引数 rec に渡して、これを呼出す。選択抽出に利用。 ※詳細は下記「DBMAN クラスプログラミング例」の「レコード検索と処理」を参照のこと ※Rel20061216 以降
recordlist():string[]	データベースオブジェクトもしくはレコードオブジェクトが保有する全レコードのキーワードを、レコード順に要素として格納した配列を返す。
search(key:string, match:bool=false):DBREC	key で指定するキーワードを含むレコードのレコードオブジェクトを返す。該当するレコードがない場合は null を返す。 match に true を指定すると、key に一致するキーワードを持つレコードを対象とする
toString():string	データベースオブジェクトの場合は、“DBMAN:＜データベースファイル名＞” レコードオブジェクトの場合は、“DBREC:＜データベースファイル名＞://＜レコードパス並び＞” を返す。レコードパス並びとはレコードキーを階層順に"/"で区切ったもの。

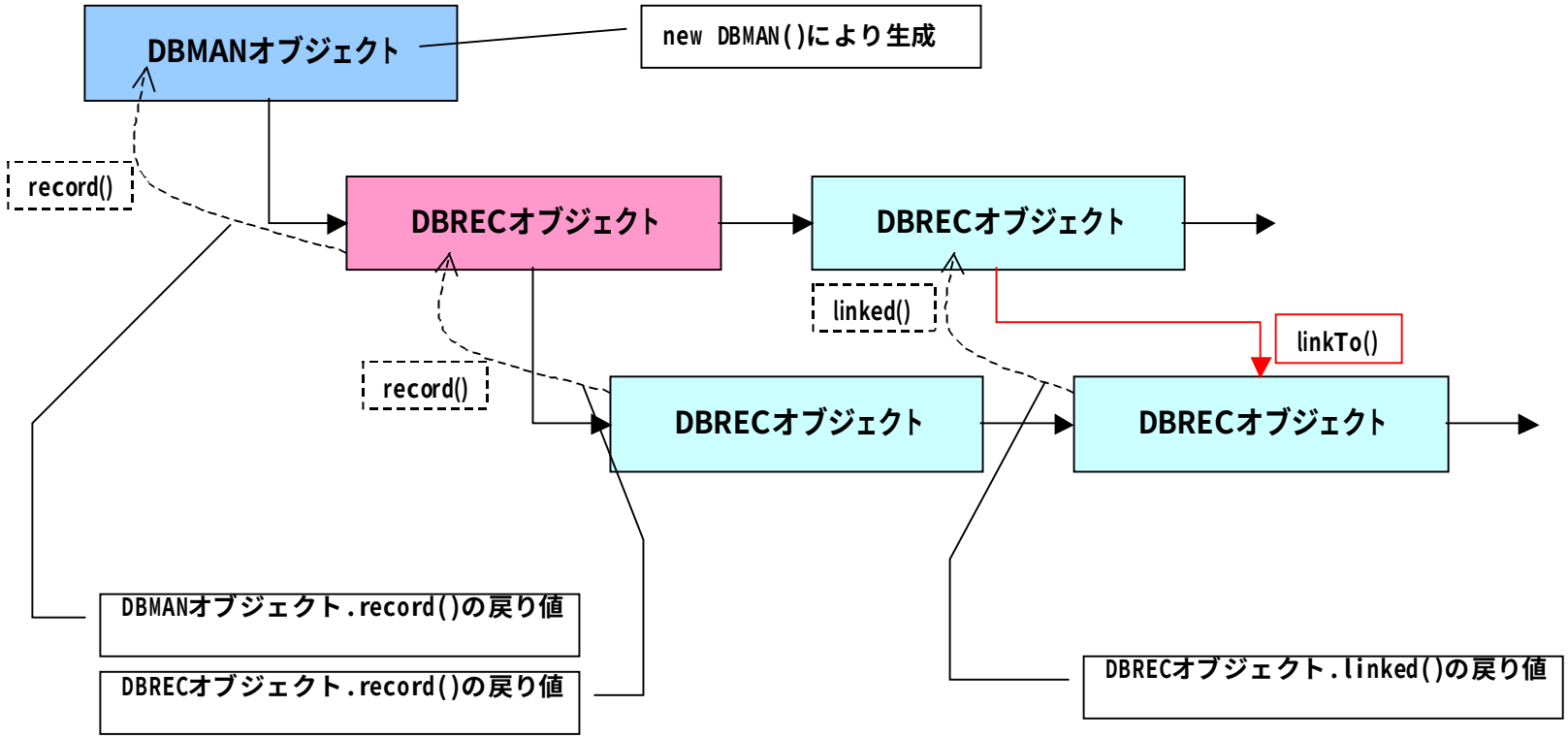
[7] レコードオフセットは整数０から始まる正整数であり、フォルダ内のレコード数を超えてはならない。

searchNext(key:string, match:bool=false):DBREC	search()ならびに searchNext()の結果得られたレコードに続くレコードのうち、key で指定するキーワードを含むレコードのレコードオブジェクトを返す。該当するレコードがない場合は null を返す。match に true を指定すると、key に一致するキーワードを持つレコードを対象とする
put(data:variant):this	data で示す値をレコードに置換格納する。成功時は自身を、失敗時は null を返す。 ※data は基本型（整数、実数、論理値、文字列、NULL）およびユーザ定義オブジェクト、ハッシュ、配列、ならびに組込みクラスの Date オブジェクト、Buffer オブジェクトのみ。
putHash(data:variant):this	data で示す値をレコードに置換格納する。成功時は自身を、失敗時は null を返す。 ※data にインスタンスを指定した場合はハッシュ化して格納する。
get():variant	レコードの内容を返す。 レコードにデータが格納されていない場合は null を返す。
getKey():string	レコードの持つキーワードを文字列で返す。
setKey(key:string):bool	key で示す文字列をキーワードとしてレコードに設定する。成功時は true を返す ※key として指定できる文字列は 64 バイト未満でなければならない。
sort(asInt:bool=false, rize:bool=false):bool	レコードのキーワード文字列をソート条件判定し、レコードの位置を決定する。 asInt に true を指定した場合は、キーワードを整数表記文字列（※）であると判断される範囲で数値化し、その結果をソート条件とする。 rize に false を指定した場合は昇順とし、true を指定した場合は降順になるようにレコード位置を決定する。成功時には true を返す。 当該レコード位置のみを決定するものであり、他のレコード位置には影響を及ぼさない。 ※結果的にレコード順序および位置（オフセット）が変わることに注意しなければならない Rel20110726 以降では、asInt に true を指定した場合、実数表記として解釈し比較する。
allocate(idx:int):bool	レコードを同じ階層内の、整数 idx (>=0) で示すレコードオフセット位置に割付ける。ただし、idx がレコード数を超えている場合は最終レコードとして割付ける。成功時には true を返す。 ※結果的にレコード順序および位置（オフセット）が変わることに注意しなければならない
getIndex():string	レコードに格納されているインデックス情報を取り出す（matchOf()参照）
setIndex(key:string):bool	key で示す文字列をインデックス情報として格納する。key に空文字列（""）を指定すると、インデックス情報を削除する。成功時には true を返す。 ※インデックス情報は matchOf()メソッドによって評価される。
putBuffer(buf:Buffer):this	Buffer オブジェクトをレコードに置換格納する。成功時は自身を、失敗時は null を返す。 レコードに大容量のバイナリデータを格納する場合は put()に比べて高速であるが、暗号化はなされない。putBuffer()によって格納したデータは、getBuffer()によって読み出さないと正しい内容とはならない。
getBuffer():Buffer	putBuffer()によって格納されている Buffer オブジェクトを読み出して返す。
remove():bool	レコード、およびそのレコードに内包されるすべての下位層レコードを削除する。成功時は true を、失敗時は false を返す。※実行後はこのレコードオブジェクトを使用してはならない。

linkTo(target:DBREC=null)	以後、このレコードに対する get(),put()は target で指定されたレコードに対してなされる。 リンク先レコードが削除された場合、および linkTo(null)実行によってリンク解除する。
linked():DBREC (Preliminary)	このレコードが linkTo()によって他のレコードにリンクしている場合､リンク先のレコードオブジェクトを返す。リンクしていない場合は null を返す。
transfer(target:DBREC):bool	レコードのデータおよび保有する下位階層レコード全体を target に移管する。正しく移管が行われた場合には true を返す。 transfer 実行以前に target が保有していたデータおよび下位階層レコードは破棄され、このレコードから移管されたデータおよび下位階層レコードに置換わる。 このレコードのデータは null となる。
updateDate():real	レコードが更新された時刻を UTC 系 1970/1/1 0:0:0 からの経過ミリ秒をあらわす実数値で返す。 レコード生成直後、put 系メソッド実行までは 0.0 である。
updateDate(msec:real):real	レコード更新時刻を UTC 系 1970/1/1 0:0:0 からの経過ミリ秒をあらわす実数値 msec で示す値に設定し、設定前の値を返す。
getCRC():int	レコードの CRC 値を返す。CRC 値は put 系メソッド実行によりデータ内容に従い更新される。
getSize():int	レコードのデータ長を返す。（putBuffer()したレコードのみ）
parent():DBREC	上位層レコードのレコードオブジェクトを返す。なければ null。
child():DBREC	下位階層レコードの先頭レコードオブジェクトを返す。なければ null。
bottom():DBREC	下位階層レコードの最終レコードオブジェクトを返す。なければ null。
elder():DBREC	同一階層レコードの直前レコードオブジェクトを返す。なければ null。
little():DBREC	同一階層レコードの直後レコードオブジェクトを返す。なければ null。

クラス・コンスタント	
DBMAN.READ:int	open()の mode 引数に指定し、読込モードを意味する。
DBMAN.WRITE:int	open()の mode 引数に指定し、読み書きモードを意味する。 自動的に排他ロックとなる。
DBMAN.CREATE:int	open()の mode 引数に指定し、読み書きモードかつ、ファイルが存在しない場合には生成する。 すでにファイルが存在する場合は DBMAN.WRITE として機能する。自動的に排他ロックとなる。
DBMAN.LOCK:int	open()の mode 引数に指定し、ファイルロックを意味する。
DBMAN.MKDIR:int	open()の mode 引数に DBMAN.CREATE とともに指定し、パス中のディレクトリが存在しない場合にこれを作成してからデータベースファイルを作成する。（※Ver4.0 以降）

21.1データベースの管理構造



データベースは、new DBMAN(＜データベース・ファイルパス＞)によって開かれ、DBMAN オブジェクトが生成される。
DBMAN オブジェクトに対して newRecord()もしくは自動生成指定による record()メソッドを適用することによって第１階層レコードを生成し、その戻り値は生成したレコードに対応した DBREC オブジェクトとなる。
DBMAN オブジェクトに record()メソッドを適用した戻り値（レコードが存在する場合のみ）は DBREC オブジェクトであり、各々第１階層のレコードに対応する。
DBMAN オブジェクトに length()メソッドを適用した場合の戻り値は第１階層レコードの総数を表す。
DBMAN オブジェクトに recordset()メソッドを適用した戻り値は、条件に適合する第１階層レコードに対応する DBREC オブジェクトの配列であり、その要素順序は実際のレコードの順序に従っている。レコードの順序は newRecord()もしくは record()にて自動生成指定した場合の「挿入・追加」指定により、また対象とするレコードに対応する DBREC オブジェクトに対して sort()メソッドを適用することによって決定される。

各々のレコードは、その内部にさらにレコードを持つことができ、結果データベースは階層構造を持ち得る。
レコード内に更にレコードを生成するには、DBREC オブジェクトに対し、newRecord()もしくは自動生成指定による record()メソッドを適用する。その戻り値は、生成した内包レコードに対応した DBREC オブジェクトとなる。
DBREC オブジェクトに対し、record()メソッドを適用した戻り値は、内包レコードの DBREC オブジェクトである。（存在する場合）
DBREC オブジェクトに length()メソッドを適用した場合の戻り値は内包レコードの総数を表す。
DBREC オブジェクトに recordset()メソッドを適用した戻り値は、条件に適合する内包レコードに対応する DBREC オブジェクトの配列であり、その要素順序は実際のレコードの順序に従っている。内包レコードの順序は DBMAN オブジェクトの場合と同様、newRecord()もしくは record()にて自動生成指定した場合の「挿入・追加」指定により、また対象とする内包レコードに対応する DBREC オブジェクトに対して sort()メソッドを適用することによって決定される。
内包レコードをもつレコードを特に、フォルダレコードというが通常のレコード同様データを保持することもができる。
フォルダレコードであるかどうかは、length()メソッドの戻り値（すなわち 0 でなければフォルダレコード）でわかる。
DBREC オブジェクトに linkTo()メソッドを適用した以降は、その DBREC オブジェクトに対する linked()メソッドは linkTO()で指定した DBREC オブジェクトを返す。

21.2DBMAN クラスのプログラミング例

•新規作成

```
var     社員名簿[] = {"田中", "大田", "山田"};
var     給与一覧[] = {250000, 280000, 340000};

var     newdb = new DBMAN("database.nlx",DBMAN.CREATE);
// ↑新しいデータベースを生成する。
newdb.newRecord( "社員" ).put( 社員名簿 );
// ↑"社員"というキーをもつレコードを生成し、データとして 社員名簿 配列を格納する。
newdb.newRecord( "給与" ).put( 給与一覧 );
// ↑"給与"というキーを持つレコードを生成し、データとして 給与一覧 配列を格納する。
newdb.close();
// ↑データベースを閉じる。
```

•既存データベースを読む

```
var     rddb = new DBMAN("database.nlx",DBMAN.READ¦DBMAN.LOCK);
// ↑先に生成したデータベースを排他ロックして開く。（排他ロックの必要がなければ DBMAN.LOCK の指定は必要ない）
foreach (rec in rddb.recordset()){
```

```
        // ↑登録されているすべてのレコードについて
        writeln(rec.get());
        // ↑レコード内容を読み出して表示する。
        }
rddb.close();
// ↑データベースを閉じる。
```

•既存データベースに階層レコードを追加する

```
var     wrdb = new DBMAN("database.nlx",DBMAN.WRITE);
// ↑先に生成したデータベースを書き込みモードで開く。（書き込みモードでは自動的に排他ロックとなる）
var     社員[] = {
        {氏名:"鈴木一郎", 性別:true},
        {氏名:"中川知美", 性別:false}
        };
// ↑格納するデータをハッシュの配列として準備する。
foreach (element in 社員){
        // ↑すべての配列要素（ハッシュ）について
        wrdb.record( "社員" ).newRecord( element.氏名, true).put(element);
        // ↑”社員”フォルダレコード内にキーが社員名の新たなレコードを追加生成して、社員配列の要素を格納する。
        // 上記ステートメントは
        //      var     社員 rec = wrdb.record( "社員" );
        //      var     newrec = 社員 rec.newRecord( element.氏名, true);
        //      newrec.put(element);
        // と同じである。
        }
wrdb.close();
// ↑データベースを閉じる。
```

•多重リンクレコードを構成する

郵便番号で示す地域について、郵便番号で検索する場合と、都道府県・市区町村・町域で検索する場合のように、同じデータに対して複数の検索アプローチをとりたい場合には多重リンクレコードを構成すると効率がよい。
多重リンクレコードはデータ自体は単一であるため、データの更新は１ヶ所でよく、また記録されるデータが複数にならないためデータベース容量も少なくできるメリットがある。
例えば `record("東京都").record("中央区").get()` **で得られるデータを、** `record("104").get()` **としてもアクセスできるようにした**

い場合を想定する。
多重リンクを構成するには リンク元レコードに対し linkTo()メソッドを適用する。
引数としてリンク先レコードを指定すると、以後の get()および put()メソッドが対象とするデータはリンク先のレコードのデータとなる。
具体的なプログラミング例としては

```
var     database:DBMAN = new DBMAN("地域.nlx",DBMAN.WRITE);          // 地域データベースを WRITE モードで開く
var     target:DBREC = database.record("東京都").record("中央区");    // "東京都"/"中央区"キーで特定されるレコード
var     links:DBREC = database.newRecord("104");                     // "104"キーで特定されるレコードを新設
links.linkTo( target );                                              // "104"レコードを"東京都"/"中央区"レコードにリンク
target.put("銀座");                                                  // リンク先レコードに"銀座"文字列を格納
writeln(links.get());                                                // リンク元でその内容が読み出せる
```

多重リンクはどのレコードに対しても、複数のレコードからリンクできる。
リンクを解消する場合は、linkTo()もしくは linkTo(null)とすれば、以後はそのレコードのデータを get()および put()する。

•レコードの検索と処理

```
var     rddb = new DBMAN("database.nlx",DBMAN.READ¦DBMAN.LOCK);
// ↑データベースを読み込みモードかつ排他ロックで開く。
foreach(rec in rddb.record( "社員" ).recordset()){
        // ↑「社員」フォルダー内のすべてのレコードについて
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。　（※レコードデータがオブジェクトの場合、参照となる）

        if(data.氏名.indexOf("中川") == 0){
        // ↑氏名項目の文字列が"中川"から始まっていれば
                var     性別:sting;
                if(data.性別) 性別 = "男"; else 性別 = "女";
                writeln(data.氏名,":",性別);
                // ↑＜氏名＞：＜性別＞書式で表示する
                }
        }
rddb.close();
// ↑データベースを閉じる。
```

レコード検索と処理を明確に分ける場合は
recordset(callback: method): DBREC[] を使用する。（Rel20061216 以降）

このメソッドは結果（ DBREC[] ）として返すレコード配列要素の選択を（場合によっては構築も）すべて callback の動作に委ねる。
引数として渡す callback メソッドの実装は、動作の違いによって二通りある。

■callback 内でレコードを選択し、レコード配列の構築のみ recordset()に委ねる場合
method callback(rec: DBREC): bool
として実装する。
recordset()は callback(rec)の結果として true が返された場合のみ、引数に渡した rec を recordset()の結果として返すべきレコード配列に追加格納する。
例：

```
method  selector(rec: DBREC): bool{
// ↑下記 recordset(selector) によってコールバックされる実装　（レコード順に呼出される）
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。　（※レコードデータがオブジェクトの場合、参照となる）
        if(data.氏名.indexOf("中川") == 0) return true;
        // ↑氏名項目の文字列が"中川"から始まっていれば 条件成立とみなして true を返す。
        }

var     rddb = new DBMAN("database.nlx",DBMAN.READ¦DBMAN.LOCK);
// ↑データベースを読み込みモードかつ排他ロックで開く。
foreach(rec in rddb.record( "社員" ).recordset(selector)){
        // ↑「社員」フォルダー内のすべてのレコードについて selector を適用した結果のレコード群について
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。　（※レコードデータがオブジェクトの場合、参照となる）
        var     性別:sting;
        if(data.性別) 性別 = "男"; else 性別 = "女";
        writeln(data.氏名,":",性別);
        // ↑＜氏名＞：＜性別＞書式で表示する
        }
rddb.close();
// ↑データベースを閉じる。
```

■callback 内で、レコード配列を参照したり自身でレコード配列に要素を追加する場合
method callback(rec: DBREC , recary[]: DBREC)
として実装する。
callback に対してはレコードごとに呼出される際に、recordset()が結果として返すレコード配列への参照が recary 引数として引き渡

される。callback 内で rec を recary の要素として追加するには recary.append(rec) とする。
例：

```
method  selector(rec: DBREC , recary[]: DBREC){
// ↑下記 recordset(selector) によってコールバックされる実装
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。　（※レコードデータがオブジェクトの場合、参照となる）
        if(data.氏名.indexOf("中川") == 0) recary.append(rec);
        // ↑氏名項目の文字列が"中川"から始まっていれば 配列に追加する。
        }

var     rddb = new DBMAN("database.nlx",DBMAN.READ¦DBMAN.LOCK);
// ↑データベースを読み込みモードかつ排他ロックで開く。
foreach(rec in rddb.record( "社員" ).recordset(selector)){
        // ↑「社員」フォルダー内のすべてのレコードについて selector を適用した結果のレコード群について
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。　（※レコードデータがオブジェクトの場合、参照となる）
        var     性別:sting;
        if(data.性別) 性別 = "男"; else 性別 = "女";
        writeln(data.氏名,":",性別);
        // ↑＜氏名＞：＜性別＞書式で表示する
        }
rddb.close();
// ↑データベースを閉じる。
```

■検索完了の判断に伴う検索処理の中止
recordset()は、あるレコードに関して呼出したコールバック内から例外（データ型を問わない）が発行された場合、残りのレコードに対する抽出とコールバック呼出しを中断し、この時点までに構成されているレコード配列を結果として返す。
コールバックは、recordset()の結果として返すべきレコード群が確定した時点で例外を発行すれば効率のよい検索ができる。

21.3メソッド呼び出しに伴うエラーと例外の発生

DBMAN.enableExcept()実行後、DBMAN.disableExcept()実行以前に、DBMAN インスタンスあるいは DBREC インスタンスのメソッド内でのエラーの発生時に RuntimeException 例外を通知する。
例外値に getValue()メソッドを適用することによって文字列によるエラー内容を得ることができる。
なお、重要なエラーについてはワーニングレベル１以上の場合 Inspect アボートする。
例：

```
DBMAN.enableExcept();
// ↑DBMAN 内での例外発行を許可
try{
        var     db = new DBMAN("userdb.nlx");
        // ↑指定したデータベースファイルがない場合、例外を発行する。
        writeln(db.length());           // 第１階層レコード総数を表示
        db.close();
}
catch(err:RuntimeException){
        // ↑RuntimeException 型オブジェクトが発行された場合に catch する。
        writeln(err.getValue());
        // ↑エラー内容を表示する
}
```

21.4保護付きデータベースのオープン

パスワードを設定してある保護付きデータベースは、open()メソッド時の psw 引数が一致した場合のみオープンできる。

21.5foreach、eachof 構文対応（Var 4 以降）

DBMAN ならびに DBREC インスタンスを foreach ならびに eachof 構文の配列参照項として利用できる。
ただし、foreach もしくは eachof 構文を nest する場合、外側構文の配列参照部に指定した同じ DBMAN および DBREC インスタンスを内側構文の配列参照部に指定してはならない。

例：

```
var     rddb = new DBMAN("database.nlx",DBMAN.READ¦DBMAN.LOCK);
foreach(rec in rddb.record( "社員" ).recordset()){
        // ↑「社員」フォルダー内のすべてのレコードについて
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。
        }
rddb.close();
// ↑データベースを閉じる。
```

では、rddb.record("社員").recordset() によって「社員」フォルダ内のすべてのレコードの DBREC インスタンスを配列要素とする配列を生成し、その配列への参照値を foreach 構文の配列参照部に引き渡しているのに対し、

```
var     rddb = new DBMAN("database.nlx",DBMAN.READ¦DBMAN.LOCK);
foreach(rec in rddb.record( "社員" )){
        // ↑「社員」フォルダー内のすべてのレコードについて
        var     data = rec.get();
        // ↑data にレコード内容を読み出す。
        }
rddb.close();
// ↑データベースを閉じる。
```

rddb.record("社員")　があたかも DBREC インスタンスの配列であるかのように振舞う。
foreach 構文ブロックの開始タイミングで順に、 rddb.record("社員")フォルダ内のレコードの DBREC インスタンスが生成され、rec を介して利用される。

22Zip クラス

ＺＩＰファイルを扱うクラス。

コンストラクタ	
Zip()	Zip オブジェクトを生成する。
Zip(path:string)	path に指定する ZIP ファイルを開いて Zip オブジェクトを生成する。

クラス・メソッド	
compress(buf:Buffer):Buffer	buf が保有するメモリーイメージを LZW 圧縮した結果を格納する、新たな Buffer オブジェクトを返す
decompress(comp:Buffer):Buffer	comp が保有する LZW 圧縮されているメモリーイメージを復号した結果を格納する、新たな Buffer オブジェクトを返す
enableExcept()	エラー時例外発生を許可する。既定値は disable 状態である。
disableExcept()	エラー時例外の発生を抑制する。

インスタンス・メソッド	
open(path:string):bool	path に指定する ZIP ファイルを開いて Zip オブジェクトに割り当てる。 開けた場合は true を返す 以前開いている Zip ファイルがあれば、これを閉じた後に開く。
isOpen():bool	Zip ファイルを開いている場合は true を返す。
getMember():string[]	Zip オブジェクトが開いている Zip ファイルに含まれるファイルユニットのパス情報を文字列配列として得る。ファイルが階層化されている場合、パス情報中のフォルダは"/"で区切られている。
get(ZipPath:string):Buffer	ZipPath に指定するパスに対応する Zip ユニットデータを、Buffer オブジェクトとして得る。指定した Zip ユニットが存在しない場合は null を返す。
getTime(ZipPath:string):real	ZipPath で指定するパスに対応する Zip ユニットのタイムスタンプを UTC 系 1970/1/1 0:0:0 からの経過ミリ秒をあらわす実数値で得る。指定した Zip ユニットが存在しない場合は null を返す。
close()	Zip オブジェクトが開いている ZIP ファイルを閉じる

22.1著作権について

Zip クラスにおいて、無償で公開されている zlib を利用しておりその著作権保有者を以下に示す。

```
(C) 1995 Jean-Loup Gailly and Mark Adler
```

22.2Zip クラスのプログラミング例

Zip クラスは ZIP ファイルから、内包するファイルユニットを取り出す用途に使用する。

"sample.zip"という ZIP ファイルに含まれるすべてのファイルを解凍するプログラムを例示する。
例：

```
var     zip:Zip = new Zip(“sample.zip”);                    // ZIP ファイルを開く
var     members[]:string = zip.getMember();                 // ZIP ファイル中のユニットのパスを配列として得る
foreach(member in members){
        var     filebody:Buffer = zip.get(member);          // ZIP ファイルパス対応のファイル内容を取り出す
        var     file:Storage = new Storage(member,          // ファイルパスと同じファイルを作成する（フォルダ対応）
                        Storage.CREATE|Storage.MKDIR);
        file.put(filebody);                                 // ファイル内容を書き出す
        file.close();
}
zip.close();
```

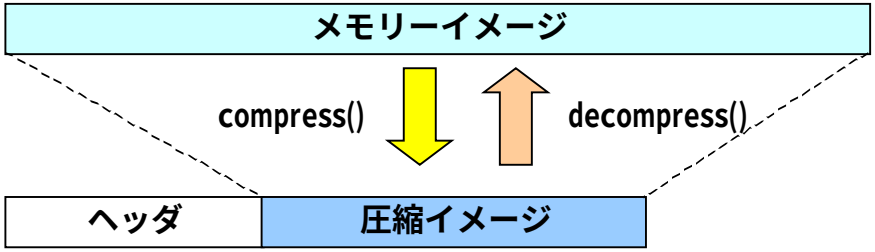
22.3メソッド呼び出しに伴う例外の発生

Zip.enableExcept()実行後から Zip.disableExcept()実行以前に、ファイルを開いていない（isOpen()の結果が false）状態で、close()、isOpen()以外のインスタンスメソッドおよびコンストラクタを実行した場合、エラー発生に伴い RuntimeException 例外を発行する。
例外値に getValue()メソッドを適用することによって文字列によるエラーメッセージを得ることができる。
例：

```
Zip.enableExcept();                             // 例外の発行を許可
try{
        var     zip = new Zip("sample.zip");    // ファイルがオープンできない場合、例外を通知する
        writeln(zip.getMember());               // ZIP ファイルがオープンできていない場合、例外を通知する
        zip.close();
}
catch(err:RuntimeException){                    // throw された例外を catch する
        writeln(err.getValue());                // エラー内容を表示する
}
```

22.4compress()、decompress()の詳細

Zip.compress()の返す Buffer オブジェクトは引数に与えた Buffer オブジェクトの保有するメモリーイメージを LZW 圧縮した内容を保有し、その構造は以下のようになっており、Zip.decompress()により復元される。compress()による圧縮と decompress()による復号は可逆的である。すなわち、ある Buffer オブジェクト buf に対し、Zip.decompress(Zip.compress(buf)) の結果返される Buffer オブジェクト内のメモリーイメージは、buf オブジェクト内のメモリーイメージとサイズおよびパターンが同一である。



compress()および decompresss()とも、引数に与えられた Buffer オブジェクトには副作用を与えない。

23Socket クラス

ソケット通信に関する機能を提供するクラス。
サーバサイドＮＡＬおよびスタンドアローンＮＡＬのみ実装

コンストラクタ	
Socket()	open()メソッドの実行を成功するまでソケットへのアクセスはすべて無効となる。
Socket(host:string, port:int:int)	引数の意味は open()メソッドと同じである。

クラス・メソッド	
hostName():string	自身のホスト名を文字列で返す。エラーなら null を返す
hostname(ipnm:string):string	ipnm で指定する IP もしくはエイリアス名に該当する実ホスト名を文字列で返す。
ipAddr(name:string):string	name で指定するホスト名の IP アドレスを"xxx.xxx.xxx.xxx"形式の文字列で返す。 name を指定しないか""の場合は、自身の IP アドレスを返す。 複数の IP アドレスを持つ場合は各々を半角スペース１文字で区切った値を返す。 （例："192.168.0.1 61.112.69.76"） エラー時は""を返す
enableExcept()	エラー時例外発生を許可する。既定値は disable 状態である。
disableExcept()	エラー時例外の発生を抑制する。

インスタンス・メソッド	
open(host:string, port:int):bool	ホストおよびポートを指定して、ソケットを接続（コネクト）する。 host にはホスト・ドメイン形式（"nal.act.ne.jp"）かＩＰ形式（"210.163.83.210"）で指定し、port は整数値を指定する。 現在すでにソケットをオープンしている場合は一旦これをクローズしてから指定されたパラメータに従ってオープンする。 オープンに成功すれば true を、失敗すれば false を返す
close()	ソケットの接続を解除し閉じる。 以後のソケット操作は open()メソッド以外すべて無効となる。
get(length:int):Buffer	length に指定するバイトサイズ分ソケットから読み出し、Buffer オブジェクトとして返す。 length を指定しない場合は既定値が用いられる。 返された Buffer オブジェクトの扱いに関しては Buffer クラスを参照のこと。 ソケットを open していない場合は null が返される。

put(value:Buffer):int	value に指定した Buffer オブジェクトをソケットに送りこむ。 送出されるデータは先頭から有効範囲までである。 結果として送出したバイト数が返される。 -1 が返される場合はエラー発生を意味する。
put(str:string):int	str に指定した文字列をソケットに送りこみ、送出したバイト数を返す。 -1 が返される場合はエラー発生を意味する。
isOpen():bool	ソケットが開いている場合（get(),put()可能）は true を返す。
server_open(port:int, maxch:int=5):bool	ソケットをサーバモードでオープンする port にはポート番号を、maxch には同時受け付け可能なクライアント数を指定する。maxch を指定しない場合は 5 を指定したものとして扱う。
server_stat(timeout:int=0):int	クライアントからのアクセス状況に応じて以下の値を返す。 timeout にm秒単位の整数を指定すると、その時間内に状態変化がない場合タイムアウトステータスを返す。指定しないか０を指定した場合は状態変化を検出するまでブロックされる。 0：アイドル状態 1：クライアントが接続した 2：クライアントが切断した 3：クライアントから書き込みがあった -1：タイムアウトもしくはエラー
server_info(id:int):int or string	id に指定する値に応じて以下の情報を返す 0 を指定した場合：クライアント識別子を整数で返す 1 を指定した場合：クライアントのＩＰ値を文字列で返す 2 を指定した場合：クライアントのポート番号を整数で返す
get(length:int,fd:int):Buffer	fd に server_info(0)によって得られたクライアント識別子を指定し、そのクライアントからのパケットを読込む。
put(value:Buffer,fd:int):int	fd に server_info(0)によって得られたクライアント識別子を指定し､そのクライアントに value で指定するデータパケットを書出す。 成功した場合は書き出したバイトサイズを、失敗した場合は-1 を返す。
put(value:string,fd:int):int	fd に server_info(0)によって得られたクライアント識別子を指定し、そのクライアントに str で指定するデータパケットを書出す。 成功した場合は書き出したバイトサイズを、失敗した場合は-1 を返す。
client_close(fd:int)	fd に server_info(0)によって得られたクライアント識別子を指定し、そのクライアントからの接続をクローズする。

※ 青字のインスタンスメソッドはサーバモード専用

23.1Socket クラスのプログラミング例

```
var     socket = new Socket();
socket.open("www.act.ne.jp", 80);
// ↑ソケットオブジェクトを生成し Web サーバに接続する（ポートは既定値の８０）

if(socket.isOpen()){
// ↑ソケット接続完了を検査

        socket.put("GET /index.html HTTP/1.0\r\n\r\n");
        // ↑サーバに HTTP リクエストを発行する

        do{
                var     buf = socket.get(1024);
                // ↑ソケットからサーバ応答を取り出す（サイズは１０２４バイト以内）

                if(buf.length() == 0)   break;
                do{
                        var     str = buf.getString();
                        if(str == "") break;
                        write(str);
                        }while(true);
                }while(true);
        socket.close();
        // ↑ソケットを閉じる
        }
else    writeln("Socket open failed!");
```

23.2メソッド呼び出しに伴う例外の発生

Socket.enableExcept()実行後から Socket.disableExcept()実行以前に、コンストラクタおよびメソッドを実行した場合、エラー発生に伴い IOException 例外を発行する。
例外値に getValue()メソッドを適用することによって文字列によるエラーメッセージを得ることができる。

23.3Socket クラスのサーバモード・プログラミング

```
var     sock = new Socket();
sock.server_open(8000);
// ↑ソケットオブジェクトを生成しポート 8000 で Listen する

repeat{
        var     sts:int = sock.server_stat(5000);
        // ↑ソケット状態を監視（タイムアウト５秒）
        select(sts){
          case  1:      writeln("connect = ",sock.server_info(1),":",sock.server_info(2));
          case  2:      writeln("disconnect = ",sock.server_info(1));
          case  3:{
                writeln("receive from ",sock.server_info(1));
                writeln(sock.get().getText());
                // ↑ソケットから受信し、内容を表示
                sock.client_close();
                // ↑サーバ側から disconnect
                }
          }
        }
```

24Document クラス

ドキュメント出力に関するクラスメソッドを提供する抽象クラス。クラス拡張はできない。
サーバサイドＮＡＬのみ実装

クラス・メソッド	
write() print()	組み込み関数 write()と同じ
writeln() println()	組み込み関数 writeln()と同じ
length():int	ドキュメントにバッファリング出力された文字列バイト長を得る。 buffer(true)以後に出力された文字列バイト長を示し、buffer(false)によって 0 となる。
buffer(sw:bool)	sw に true を指定すると、以後の標準出力をバッファリングする。false を指定するとバッファリングされた標準出力をフラッシュし、以後の標準出力をバッファリングしない。
avoid()	バッファリングされている標準出力を破棄する
pop():string	バッファリングされている標準出力を文字列として取り出し、バッファを破棄する。 結果 length()は 0 となる。
flush()	バッファリングされた標準出力をフラッシュする。 結果 length()は 0 となる。

別名として class document でもよい。
例： **Document**.writeln() は **document**.writeln()　と同じ。
すなわち、writeln()、Document.writeln()、document.writeln()、Util.writeln()は同じである。

24.1buffer メソッドの使い方

初期状態では write()および writeln()、さらにパス文（%%文字列%%）およびタグ文（<A href="http://..">等）はそのまま標準出力に対して出力される。Document.buffer(**true**)を実行した後は、Document.buffer(**false**)を実行するまで上記出力はバッファリングされ、Document.buffer(false)の呼出しあるいはプログラムの終了によってバッファリングされている内容が標準出力に出力される。Document.buffer(false)実行以前に Document.pop()を実行すると、バッファリングされている内容を文字列として取り出すとともに、バッファは空となる。

以下の例はパス文で囲まれたソースコードを表示するとともに実行する。

```
Documen.buffer(true);
%%writeln(“ I am a boy ”);%%                    // "writeln("I am a boy");"  がバッファリングされる。
var     source:string = Document.pop();
Documen.buffer(false);
```

```
writeln(source);                    // "writeln("I am a boy");" を表示する
eval(source);                       // "I am a boy" を表示する
```

25Util クラス

Util クラスはＮＡＬの組み込み関数をクラスメソッドとして提供する抽象クラスであり、クラス拡張はできない。
各々の関数仕様については、「ＮＡＬプログラミング・リファレンス」の組み込み関数の項を参照のこと。

25.1Util クラスの使用例

組み込み関数　writeln()を使用する場合、Util.writeln()と記述しても良い。
プログラム内で writeln を再定義した場合、組み込み関数としての writeln()は隠蔽されるが、Util.writeln()は正しく組み込み関数を呼び出すことができる。但し、Util を再定義した場合は適用できない。

例：

```
var     writeln = “Override define”;          // writeln を再定義
Util.writeln(writeln);                        // 組み込み関数の writeln()を呼び出す
```

26System クラス

クラスメソッドのみ提供する抽象クラス。クラス拡張はできない。

クラス・メソッド	
exec(command:string):int	文字列 command で指定するプログラムを実行し、そのプロセスＩＤ（pid）を返す。 プログラムの実行完了を待たないため、プログラムの終了を待つには System.wait()すること。
pipe(command:string):Buffer	文字列 command で指定するプログラムを実行し、プログラムの実行完了を待ってその標準出力を Buffer オブジェクトとして返す。
getpid():int	NAL の pid 値を返す（pid は OS によって既定された値である）
wait(pid:int):int	pid で指定する子プロセス（System.exec()の結果）の終了を待つ。 結果として子プロセスの終了コードを返す。
wait(pid:int,true):bool	pid で指定する子プロセスの終了を検査する。終了していない場合は true を返す。※Linux のみ
log(式並び)	式並びの値をログファイルに出力する
logpath(path:string):string	ログファイルのパスを path で指定し前回値を返す。（既定値は"nal.log"） 以後の System.log()にて書き出されるログは、指定したファイルに対して追記される
worning(val:int=null):int	val に指定する整数値を worning レベルとして設定し、設定以前の値を返す。val を指定しないか整数以外を指定した場合は worning レベルを変更しない。
inspect(msg:string="", asworn:bool=false)	msg を Inspect 内容として表示し、実行を停止する。 asworn に true を指定した場合は Worning 表示となり、実行は継続される。
getVersion():string	以下の文字列にて NAL のバージョン情報を返す。 "NAL/<version>/<release>/<model>/<osid>/<generate> {/<ライセンス名>}" version：実数表記　（ex: 1.0） release：YYYYMMDD　（ex: 19990219） model：ENTERPRISE or INTRANET or PERSONAL osid：Windows or Unix generate：[month DD YYYY]　（ex: [Feb 19 1999]）
getLicence():string	ライセンス名文字列を得る、ライセスが指定されていない場合は空文字列

※ System クラスは、別名で SYSTEM クラスとしても使用可能。　（例： System.getVersion() == SYSTEM.getVersion()）
※ Windows 環境では exec(),pipe(),getpid(),wait()はネットワークユーザから利用できない。

27URI クラス

class URI の代わりに class URL としても利用できる。
サーバサイドＮＡＬのみ実装

コンストラクタ	
URI()	URI インスタンスを生成する。 インスタンスが保持する port 値は-1 となり、その他の要素プロパティ値は""となる。
URI(uri:string)	uri に指定する URI 文字列をもとに要素分解したインスタンスを生成する。 例：new URI("http://www.act.ne.jp/nal/sample.nal?rqst=view&mode=0"); とすると 各々の要素を取り出す以下のメソッドに対応して、 protocol()メソッドは"http"を返す host()メソッドは"www.act.ne.jp"を返す port()メソッドは-1 を返す path()メソッドは"/nal/sample.nal"を返す query()メソッドは"rqst=view&mode=0"を返す

クラス・メソッド	
encode(str:string):string	str の文字列を URL_encode した結果を返す
decode(str:string):string	str 文字列を URL_decode した結果を返す

インスタンス・メソッド	
protocol(prot:string):string	prot に文字列を指定すればその値をプロトコルとして設定する。 結果として、設定した値もしくは現在のプロトコル値を返す。
host(hostname:string):string	hostname に文字列を指定すれば、その値をホスト名として設定する。 結果として、設定した値もしくは現在のホスト名を返す。
port(portno:int):int	portno に-1 を指定するとプロトコルの既定ポート番号となり、この値は toString()メソッド呼び出し時に無視される。整数を指定するとその値をポート番号として設定する。 結果として、設定した値もしくは現在のポート番号を返す。
path(pathstr:string):string	pathstr に文字列を指定すればその値をファイルパスとして設定する。ファイルパスは absolute 指定でなければならない。 結果として、設定した値もしくは現在のファイルパス値を返す。
query(querystr:string):string	querystr に文字列を指定すればその値をクエリー値として設定する。 結果として、設定した値もしくは現在のクエリー値を返す。
toString():string	現在の各プロパティ値をもとに URL 文字列を生成して返す。

27.1URI クラスの利用方法

URI 文字列を指定してインスタンスを生成する用法では、URI 要素を分解した結果を取り出す際に使うとよいが、また各々の要素値を設定して対応する URI 文字列を得るといった用途にも利用できる。

URI.encode()、URI.decode()はダイナミックドキュメント生成時に、hidden クエリーをクライアントに渡して、これを再び引数として返してもらう場合に、一般のクエリー文字列と識別する目的で使用する。すなわち、あるデータ群をクエリーとして解析されないようにする仕組みと考えれば良い。

•ページ生成での使用例：

```
var     arg = {arg1:"?abc=123", arg2:100, arg3:"&xyz=ABC&stu=1234"};
<input type="hidden" name="arg" value="%{ URI.encode(Value.serialize(arg)) }%" />;
```

•クエリー解析での使用例：

```
var     qarg = Value.eval(URI.decode(arg));
writeln(qarg.arg3);                                    ①
```

上記例では、ページ生成にてハッシュ変数 arg の内容をそのままクライアントに送りこみ、クライアントから submit によってクエリーarg として受け取った後、その内容をハッシュに復元した後、arg3 キーの内容を表示している。
①の結果から ”&xyz=ABC&stu=1234” がクライアントを経由しても保存されていることがわかる。
なお、Value.serialize()と Value.eval()はハッシュを文字列化（シリアライズ）したりまたその逆変換に使用しており、扱うデータが文字列である場合は導入する必要はない。

※URI.encode()と URI.decode()は対で使用されるべきであり、あるデータに対する操作回数が異なった場合は、そのデータは保全されないことに注意すること。

28Query オブジェクト

サーバサイドＮＡＬのみ実装

クライアント（ブラウザ）からのクエリー文字列を保有し、その値を取り出すメソッドのみ提供する Wrapper オブジェクトである。

メソッド	
getHash():object	Query 文字列を URL デコードした後、キーセットに分解したハッシュとして返す。 同名のフォーム値（例：<SELECT name=″＜key＞″ multiple> タグ）に複数の値が渡されるケースではこの値をを別々に取り出すことはできず、最後の値のみが取り出される。
getParameter(key:string):string getParameter(key:string,true) :string[]	key で指定するキーに対応するパラメータ文字列を返す。対応するキーがない場合は null を返す。 第２引数に true を指定すると結果を配列に格納して返すが、対応するキーがない場合は空の配列を返し、複数の同一キーがある場合は順に要素に格納して返す。 ※配列で返す形式の呼び出しは、同名のキーセットが複数送られてくるケース （例えば <SELECT name=″＜key＞″ multiple> タグの場合など）で使用する。
getValue(dec:bool=false):string	Query 文字列全体を返す。dec に true を指定すると URL デコードした結果を返す。

28.1<FORM method=”POST” enctype=”multipart/form-data”>のフォーム値取得

<FORM method=″post″ enctype=″multipart/form-data″>は、画像などのバイナリを<INPUT type=″FILE″>でサーバに送信する場合に常套的に用いられるが、Query.getValue()や Query.getParameter(key:string)を用いて<INPUT type="FILE">に関するフォームデータを取り出すことはできない。
この場合は Query.getHash()を用いて、ハッシュ化したフォーム値オブジェクトを取得し、要素名を指定してアクセスする。
例：

```
var    form = Query.getHash();
//■ <INPUT type="FILE" name="image" />で送信されたファイルパスを得るには
var    path:string = form["image"].name;

//■ <INPUT type="FILE" name="image" />で送信されたバイナリを得るには
var    buf:Buffer = form["image"].body;
```

のように使用する。
また、<INPUT type="FILE">以外のフォーム値も、<SELECT multiple>での同一キーによる複数値を除き getHash()で取り出せる。

29Head オブジェクト

サーバサイドＮＡＬのみ実装

クライアント（ブラウザ）に引き渡すレスポンスヘッダを管理するメソッドのみ提供する Wrapper オブジェクトである。

メソッド	
write():bool output():bool	レスポンスヘッダを出力する。出力した場合は true を返し、すでに自動出力されていた場合は false を返す。 ※レスポンスヘッダは最初にクライアントに送出する命令（例：write()）あるいはタグ文によって自動送出されるが、自動送出によらず強制的にレスポンスヘッダを出力する場合に使用する。
getValue():string	レスポンスヘッダ文字列を返す。 ※Head.write()を実行したか、ドキュメント出力によるレスポンスヘッダ自動出力によってすでに出力されている場合は空文字列を返すため、出力結果を得るには Head.archive()を利用すること。
putValue(str:string)	レスポンスヘッダとして出力する文字列を str に置換設定する。 すでにレスポンスヘッダが出力されている場合は機能しない。
append(str:string,nl:bool=true)	レスポンスヘッダとして出力する文字列に str を追加する。 すでにレスポンスヘッダが出力されている場合は機能しない。 nl に true を指定した場合および無指定時は改行コードを先導挿入して追加し、false を指定した場合は改行コードを先導挿入しない。 ※レスポンスヘッダキーワードは行頭から始まるため、通常は nl=true とする。
archive():string	レスポンスヘッダおよびドキュメントの一部として自動出力された文字列を返す。レスポンスヘッダ出力以前であれば空文字列を返す。

29.1レスポンスヘッダ文字列既定値

既定値として以下の内容を持つ。

■標準

```
Status: 200 OK
Content-Type: text/html; charset=Shift_JIS
Content-Language: ja
Pragma: no-cache
Cache-control: no-cache
X-Server-ID: NAL
```

■スクリプトが EUC-JP でエンコードされているか　#charset="EUC-JP"　を指定した場合。

```
Status: 200 OK
Content-Type: text/html; charset=EUC-JP
Content-Language: ja
Pragma: no-cache
Cache-control: no-cache
X-Server-ID: NAL
```

29.2Head クラスの利用方法

サーバプログラムの最初の出力以前に Head.putValue()によって文字列を設定すると、既定値として用意しているレスポンスヘッダ文字列（以下レスポンスという）を書きかえることができる。
例：

```
Head.putValue( "Content-type: text/plain" );
writeln( "<body bgcolor=blue>" );
```

この例は、ブラウザ上に"<body bgcolor=blue>"という文字列を表示する。
すなわち、既定値のレスポンスは"Content-type: text/html"となっており、通常ＨＴＭＬドキュメントを出力する。
これを他の Content-type にするために上記例のような手法を用いる。
また新たに代入した、もしくは既定値としてのレスポンスにさらに付加情報を追加するには
Head.append(“Status: 401”);
のように、文字列の連結をおこなう。

•コンテントタイプの指定

Head.putValue("Content-type: image/gif")
などによってＨＴＭＬ以外のドキュメントをクライアントに返すことができる。

•サーバリダイレクションの指定

Head.putValue("Location: http://www.act.ne.jp/")
は、アクト社のホームページを表示する。

•ステータスコードを変更する

Head.append("Status: 401") あるいは　Head.putValue(Head.getValue().replace("Status: 200 OK","Status: 401"))
は、クライアントに Basic 認証を求める。これはサーバの認証機構を使用しないで独自認証をおこなう場合に利用する。

30Stream オブジェクト

サーバサイドＮＡＬのみ実装

STDIN、STDOUT はシステム内において各々唯一のオブジェクトである。

メソッド	
STDIN.get(length:int=null):Buffer	標準入力 STDIN のカレント以降の length バイトのデータを、Buffer オブジェクトとして返し、データがない場合は整数０を返す。 length に指定した値が EOF ポイントを超えている場合は EOF ポイントまでのバイトデータを読み込む。 length を指定しない場合は STDIN のカレント以降すべてを読み込む。 STDIN のカレントは読み込んだバイトサイズ分進む。 Buffer オブジェクトの length()値は実際に読み出したサイズを示す。 ※起動以前にリダイレクトなどによって標準入力にあらかじめ入力されているデータのみが読込みの対象となる。コンソールからの読込みをおこなう場合は readLine()メソッドを利用すること。
STDIN.length():int	標準入力 STDIN のカレントから EOF までのバイトサイズを返す。 この値は get()メソッドによって更新し、EOF ポイントに達している場合は０を返す。
STDIN.open(name:string):bool	name に指定したファイル（もしくは I/O）を標準入力として開く。 成功した場合は true を返す。
STDIN.readLine(length:int=1):string	標準入力 STDIN のカレント以降の length バイトもしくは改行文字までのデータを、文字列として返す。length を指定しない場合は 1 を指定したものとして扱う。 STDIN のカレントは読み込んだバイトサイズ分進む。
STDOUT.open(name:string, mode:string=”a+”):bool	name に指定したファイル（もしくは I/O）を標準出力として開く。 成功した場合は true を返す。 mode には”w+”,”a+”を指定することができ、”w+”を指定した場合は指定ファイルの内容を置き換える。mode を指定しない場合は”a+”が既定値として採用される。
STDOUT.put(buffer:Buffer):int	buffer で示す Buffer オブジェクトの内容を標準出力 STDOUT に書き込む。 書き込みに成功した場合はそのサイズを返し、失敗した場合および引数が正しくない場合、Buffer に有効データがなかった場合は０を返す。
STDOUT.put(text:string):int	text で示す文字列を標準出力 STDOUT に書き込む。 書き込みに成功した場合はそのバイトサイズを返し、失敗した場合は０を返す。
STDOUT.flush()	STDOUT をフラッシュする。 write()などによる STDOUT への書き出しはバッファリングされており、パケット分がそろうまではクライアントに送出されないが、フラッシュをおこなうと直ちに送出される。 ※Linux のみ

30.1プラットホーム固有の標準入出力を扱う。

•STDIN オブジェクト

スタンドアローン・モデルにおいてのみ標準入力を扱う。

•STDOUT オブジェクト

サーバサイドモデルでは、STDOUT はドキュメント出力専用のオブジェクトとして実装されている。
writeln()（実際は document.writeln()）をおこなった場合、document オブジェクトの指定がバッファＯＮになっていなければそのまま STDOUT に送出される。
STDOUT に対する最初の出力に先立って、Head オブジェクトの保有するレスポンス・ヘッダ文字列が先に送出される。

30.2Stream の使い方

クライアントからのフォーム送信において Enctype="multipart/form-data" 属性で CGI に送る場合、Ver 3.0 以前の NAL はクエリーを解析しない。この場合は、CGI プログラムにおいて STDIN.get()を使用してクライアントから与えられたデータを解析し、その処理を行なわなければならない。多くの場合、 Enctype="mutipart/form-data" 属性はファイルのアップロードに使用される。
このケースでは getenv("CONTENT_LENGTH")と STDIN.length()は表記上同じ値を示しているはずである。ただし、getenv()で得られる値は文字列であることを忘れてはならない。
デコーディング方法など詳細は RFC1867 を参照のこと。
※ ServerSide NAL Ver3.0 以降では Enctype="multipart/form-data" にも対応している。

STDOUT.put()は、イメージファイルなどをコンテンツデータとして送り出すような場合に有効である。

31Event クラス

クライアントサイドＮＡＬのみ実装

クラスフィールドのみ提供する抽象クラス。クラス拡張はできない。

クラス・フィールド	
Event.Rewrite:int	再描画イベント値
Event.Mouse:int	マウスイベント値

Event クラスの利用方法

Event クラスは、クラス定数のみ提供し、コンストラクタおよびメソッドを持たない。
一般に、イベントハンドラ関数内にて、発生したイベントの種別を判定する目的で使用する。
例：

```
function event_handler(event){
    select(event){
        case    Event.Rewrite:  再描画イベント対応処理;
        case    Event.Mouse:    マウスイベント対応処理;
        }
}
```

32Thread クラス

クライアントサイドＮＡＬのみ実装

Thread を扱うクラス

コンストラクタ	
Thread(entry:method)	entry に指定するメソッドを実行プログラムとする、スレッドオブジェクトを得る

インスタンス・メソッド	
Run()	スレッドを起動する
Abort()	スレッドの実行を終了する
sleep(sec:num=0)	sec に指定する秒間スレッドの実行を中断する。例：sleep(1.5)
waitEvent(event:int)	event で指定するイベントの発生まで実行を中断する
postEvent(event:int)	スレッドに event で指定するイベントを通知する
isExit():bool	スレッドが終了していれば true、実行中であれば false を返す
waitExit()	スレッドが終了するまで待ち合わす

Thread は、目的に応じてカスタマイズされるスレッドプログラムの、基本クラスとして使用する。

例：

```
class userThread() extends Thread{
        method  entry(){
                for(;;){
                        ステートメント;
                }
        }

        super(entry);
        Run();
}

var     usr = new userThread();
```

33Image クラス

クライアントサイドＮＡＬのみ実装
画像を扱うクラス

コンストラクタ	
Image(URL:string)	URL で示すイメージファイルをロードし、Image オブジェクトを生成する
Image(x:int,y:int,URL:string)	幅 x、高さ y のイメージオブジェクトを生成する。 URL を指定した場合はイメージファイルをロードする。 x,y を明確に指定した場合は、実際のイメージファイルの幅と高さに左右されない。

インスタンス・メソッド	
getWidth():int	イメージの幅を返す
getHeight():int	イメージの高さを返す
setWidth(ival:int)	整数 ival をイメージの幅とする
setHeight(ival:int)	整数 ival をイメージの高さとする

インスタンス・プロパティ	
src:int	イメージファイルの URL を取り出し、書き込むとそのイメージファイルをロードする
width:int	イメージの幅を取り出し、書き込むとイメージの幅を規定する。
height:int	イメージの高さを取り出し、書き込むとイメージの高さを規定する。