MITSUBISHI 三菱Web地理情報システム構築パッケージ

三菱マッピングシステム PreSerV WebTcl

PreSerV WebTcl マニュアル(Tcl コマンドリファレンス)

Ver.6.0

マニュアル

はじめに

Tcl コマンドリファレンスは１章の節ごとに１コマンドのリファレンスを記述しています。２章は expr コマンド内で使用可能な数学関数のリファレンスです。

① 目次の使い方

調べたいコマンドは目次から調べて、目次のリンクをクリックするとコマンドのページにジャンプします（PDF のリンク機能を使用しています。）。

② 使用例のファイル名について

使用例の抜粋対象としてかかれているファイル名はサンプルなどの製品付属のファイル名です。前後の処理を確認したい場合は、製品付属のファイルを参照してください。

③ 括弧の表記に関して

本マニュアルでは説明文中の括弧に関して以下の表記を行っています。
それぞれの括弧の開始は、語尾に"開き"をつけ、ワードを鍵括弧"「」"でくくります。
それぞれの括弧の終了は、語尾に"閉じ"を付けて、ワードを鍵括弧"「」"でくくります。

(a) 小括弧

（ ）を示します。形状から丸括弧と呼ぶこともありますが、本書では小括弧と表記します。

(b) 中括弧

｛ ｝を示します。形状から波括弧と呼ぶこともありますが、本書では中括弧と表記します。

(c) 大括弧

［ ］を示します。形状から角括弧と呼ぶこともありますが、本書では大括弧と表記します。

目次

1 Tcl コマンドリファレンス

Tcl インタプリタの組み込みコマンドのコマンドリファレンスを以下で説明します。
コマンドはアルファベット順に並んでいます。
※使用例は、コーリング形式が非常に単純なものや、説明中で使用例を提示しているものは記述を省略しています。

1.1 append

append	変数に値を追加します
形式	append varName value [value value ...]

説明

変数 varName の値に、全ての value 引数の値を追加します。

varName 新規の変数の場合、全ての value 引数をつなげた値を代入します。

このコマンドは長い値を持つ変数を、何度かに分けて作成する時に効率的です。

たとえば “append a $b” は $a が長い時には “set a ab” より遥かに効率的です。

使用例

#charlist.psv から抜粋。漢字コードバイナリ用の文字列をループで生成する処理

```
for {set i $ssb} {$i <= $seb} {incr i } {
        set strlist ""
        for {set j 0} {$j <= 15} {incr j} {
                set lmb [expr $i * 16 + $j]
                if {$lmb == [expr 0x7f]} {
                        set val [expr 0x81]
                        append val " " [expr 0x40]
                } elseif {$lmb > [expr 0xfc]} {
                        set val [expr 0x81]
                        append val " " [expr 0x40]
                } else {
                        set val $hmb
                        append val " " $lmb
                }
                append strlist $val " " [expr 0x81] " " [expr 0x40] " "
        }
        set binlist [binary format c* $strlist]
```

1.2 array

array	配列変数に関する操作を行います
形式	array option arrayName [arg arg ...]

説明

このコマンドは arrayName で指定する変数に対して、option で指定する操作を行います。arrayName は存在する配列変数だけが指定可能です。option に指定できる値を以下に例示します。

array anymore arrayName searchId

配列探索でまだ処理していない要素があれば 1 を、もうなければ 0 を返します。

searchId は arrayName に対するどの探索かを示すもので、 array startsearch によって返す値を指定します。このオプションは空の名前を持つ要素を持った配列に対して有効です。

これは array nextelement の返す値からは探索が終ったことが判断できないためです。

array donesearch arrayName searchId

このコマンドは配列探索を終わらせ、その探索に関する全ての状態情報をクリアします。

searchId には arrayName に対するどの探索を終わらせるのかを指定します。

これは array startsearch によって返す値でなければなりません。このコマンドは空文字列を返します。

array names arrayName

配列の全要素の名前から構成されるリストを返します。もしも一つも要素がなければ空文字列を返します。

array nextelement arrayName searchId

arrayName の次の要素、あるいは arrayName の全ての要素がすでにこの探索で返している場合には空文字列を返します。searchId 引数は探索の指定で、これは array startsearch によって返した値でなければなりません。

注）もし配列に要素を追加した時や消去した時には、全ての探索は、array donesearch が実行したように自動的に終了してしまいます。したがって array nextelement は失敗します。

array size arrayName

配列の要素の数を返します。

array startsearch arrayName

このコマンドは arrayName で指定する配列の要素ごとの探索を初期化し、array nextelement コマンドで配列の個々の要素名が返すようにします。

探索が終ったら array donesearch コマンドを呼ぶ必要があります。

このコマンドは array nextelement や array donesearch コマンドで使用するための探索の識別子を返します。

識別子とコマンドの使用により、同一の配列に対して同時に複数の探索を行うことが可能です。

使用例

```
# st_op.psv から抜粋。stval の要素名のリストを取得し、stids 要素が無い場合は初期化する処理
if { [lsearch [array names stval] stids] < 0 } {
        set stval(stids) ""
}
```

1.3 binary

binary	バイナリ文字値と通常文字列との入出力変換操作を行います
形式	binary format formatString arg [arg ...] binary scan string formatString varName [varName ...]

説明

このコマンドはバイナリデータの操作をおこないます。

まず、"binary format"の形式では通常の文字列形式の変数値(arg)からバイナリの変数値(戻り値)を作成します。

たとえば、"16 22"という値から、4byte の整数値を２つ組み合わせた 8bytes のバイナリ変数値を作成します。

次に、"binary scan"の形式では、逆にバイナリ変数値(string)から文字列の変数値のリスト(varName)を抽出します。

"binary format"コマンドでは formatString とあとに続いて指定する実際の数値部分から、バイナリ文字列を生成します。変換結果としてバイナリ変数値を戻り値として返します。formatString は複数のフィールド指定子で記述します。各フィールド指定子は１文字の型宣言文字とそのあとに続ける数値(count)で記述します。

ほとんどのフィールド指定子は引数 arg を消費しながら変換結果を格納していきます。

型宣言文字は、バイナリ値をどのようにフォーマットするのかを指定します。

あとに続く数値は普通指定した型のアイテムを何回繰り返すかを指定します。

通常は数値部分には正の 10 進整数値を指定しますが、すべてのアイテムに型宣言文字を適用する場合は"*"を使用します。

引数 arg の数が、数値部分の数と合わない場合、エラーになります。

各フィールド指定子の型宣言文字の変換を実行すると、変換したバイト分バイナリ変数上を、イメージカーソルが移動します。

イメージカーソルは、最初はバイナリ変数値の先頭０にあるが、型宣言文字に適合するバイト長分解析を行い、その解析後の位置にカーソルを移動させます。

型宣言文字"X"と"@"はこのカーソルに対する操作を可能にします。

型宣言文字は以下に示す文字が使用可能です。

a :

長さ count の文字列として変換します。引数 arg が count バイト以下である場合は、残りのバイト数を NULL でパティングします。引数 arg が count バイトより長い場合は、余分な文字は無視します。count が"*"の時は、args すべてのバイトが対象となります。

たとえば、

binary format a7a*a alpha bravo charlie

の変換結果は、"alpha¥x00¥x00bravoc"というバイナリ変数値を返します。

A

この形式は隙間を埋めるのに NULL を使用せずスペースコードを使用する動作を除いては"a"と同じです。

たとえば、

binary format A6A*A alpha bravo charlie

の変換結果は"alpha bravoc"というバイナリ変数値を返します。

説明

b

count 個の２進数形式の文字列をバイト単位で下位ビット～上位ビットの順に格納します。引数 arg には"1"と"0"しか指定できません。引数 arg が count バイト以下の場合は、残りのビットはすべて OFF となります。引数 arg が count バイトより長い場合は、余分な文字は無視します。count が"*"の時は、arg の数字すべてが変換対象となります。count が省略されると、１文字の数字だけが変換対象となります。変換最後のビットがバイト境界の最後のビットでない場合、最後のビット以降のビットはすべて OFF となります。

たとえば、

binary format b5b* 11100 111000011010

の変換結果は"¥x07¥x87¥x05"というバイナリ変数値を返します。

B

この形式はビットが上位ビット～下位ビットに格納される動作を除いては"b"と同じです。

たとえば、

binary format B5B* 11100 111000011010

の変換結果は"¥xe0¥xe1¥xa0"というバイナリ変数値を返します。

h

count 個の１６進数形式の文字列をバイト単位に下位４ビット～上位４ビットの順に格納します。引数 arg には、'0123456789abcdefABCDEF'内のキャラクタしか指定できません。引数 arg が count バイト以下の場合は残りのビットはすべて OFF となります。引数 arg が count バイトより長い場合は、余分な文字は無視します。count が"*"の時は、arg の英数字すべてが変換対象となります。count を省略すると、１文字の英数字だけが変換対象となります。変換最後の４ビットがバイト境界の最後のビットでない場合､最後のビット以降のビットはすべて OFF となります。

たとえば、

binary format h3h* AB def

の変換結果は"¥xba¥xed¥x0f"というバイナリ変数値を返します。

H

この形式はビットが上位４ビット～下位４ビットの順に格納する動作を除いては"h"と同じです。

たとえば、

binary format H3H* ab DEF

の変換結果は"¥xab¥xde¥xf0"というバイナリ変数値を返します。

c

１個以上の整数値をバイト単位でバイナリ変数値に格納します。count を指定しない時は、arg は１個の整数値となります｡またargはcount個以上の整数値のリストを指定してください｡変換時は各整数値の下位8bitをunsigned char 型の変数としてバイナリ変数値に格納します。count が"*"の時はリスト内の全ての整数を unsigned char 型の変数に整形します。リストの要素の数が、count よりも少ないとエラーになります。リストの要素の数が、count よりも多いと、余分な要素を無視します。

説明

たとえば、

binary format c3cc* {3 -3 128 1} 257 {2 5}

の変換結果は"¥x03¥xfd¥x80¥x01¥x02¥x05"というバイナリ変数値を返します。

また、

binary format c {2 5}

を実行すると、エラーになります。

s

この形式は１個以上の16bit整数値をリトルエディシャンのバイト順にバイナリ変数値を格納する動作を除いては"c"と同じです。各整数(arg)の下位16ビットを２バイト整数値として変換します。

たとえば、

binary format s3 {3 -3 258 1}

の変換結果は"¥x03¥x00¥xfd¥xff¥x02¥x01"というバイナリ変数値を返します。

S

この形式は１個以上の16bit整数地をビッグエディシャンのバイト順にバイナリ変数値を格納する動作を除いては"c"と同じです。

たとえば、

binary format S3 {3 -3 258 1}

の変換結果は"¥x00¥x03¥xff¥xfd¥x01¥x02"というバイナリ変数値を返します。

i

この形式では１個以上の32bit整数値をリトルエディシャンのバイト順にバイナリ変数値を格納する動作を除いては、"c"と同じです。各整数(arg)の下位32bitを４バイト整数値として変換します。

たとえば、

binary format i3 {3 -3 65536 1}

変換結果は "¥x03¥x00¥x00¥x00¥xfd¥xff¥xff¥xff¥x00¥x00¥x10¥x00"というバイナリ変数値を返します。

l

この形式は１個以上の 32it 整数値をビッグエディシャンのバイト順にバイナリ変数値を格納する動作を除いては"c"と同じです。

たとえば、

binary format l3 {3 -3 65536 1}

の変換結果は "¥x00¥x00¥x00¥x03¥xff¥xff¥xff¥xfd¥x00¥x10¥x00¥x00"というバイナリ変数値を返します。

f

この形式は１個以上のマシンネイティブな単精度浮動小数点値をバイナリ変数値に格納します。このフォーマットでは機種アーキテクチャ間での互換性はありません。このため、ネットワーク通信のデータにこのフィールド指定子を使用すべきではありません。１つの浮動小数点のサイズはアーキテクチャ間で異なります。

変換対象の値がアーキテクチャで定義する値の範囲を越えると、システムで定義するFLT_MINかFLT_MAXを代入します。

説明

また、Tcl インタプリタは倍精度の浮動小数点を内部的に使用しているので、単精度と倍精度間の変換で若干の誤差が発生する可能性があります。

たとえば、x86 系の CPU で動いている Windows システムでは、

binary format f2 {1.6 3.4}

の変換結果は"¥xcd¥xcc¥xcc¥x3f¥x9a¥x99¥x59¥x40"というバイナリ変数値を返します。

d

この形式は１個以上のマシンネイティブな倍精度浮動小数点値をバイナリ変数値に格納する動作を除いては、"f"と同じです。

たとえば、x86 系の CPU で動いている Windows システムでは、

binary format d1 {1.6}

の変換結果は "¥x9a¥x99¥x99¥x99¥x99¥x99¥xf9¥x3f"というバイナリ変数値を返します。

x

count 個の NULL("¥x00")をバイナリ変数値に格納します。count が指定しない時は、１個の NULL を格納します。count が"*"の時は、エラーになります。このフィールド指定子は、引数の値を上書きせず NULL でパティングします。

たとえば、

binary format a3xa3x2a3 abc def ghi

の変換結果は "abc¥x00def¥x00¥x00ghi"というバイナリ変数値を返します。

X

出力中のバイナリ文字列のカーソルを count バイト元に戻します。count が"*"か現在のカーソル位置より大きいと、カーソルは先頭である位置０に戻します。count を省略すると、カーソルは１バイト戻ります。このフィールド指定子は、引数を消費しません。

たとえば、

binary format a3X*a3X2a3 abc def ghi

は"dghi"というバイナリ変数値を返します。

@

出力中のバイナリ変数値のカーソルを count で指定するバイト位置に移動します。count が０の時はバイナリ変数値の先頭を意味します。count で指定するバイトより大きい場合は、現在の最後以降に NULL をパティングし、count で指定する位置までカーソルを移動します。count を省略すると、エラーになります。このフィールド指定子は、引数を消費しません。

たとえば、

binary format a5@2a1@*a3@10a1 abcde f ghi j

は"abfdeghi¥x00¥x00j"というバイナリ変数値を返します。

使用例

#charlist.psv から抜粋。整数値のリストをバイナリ配列とした変数に変換する処理

```
set binlist [binary format c* $strlist]
```

1.4 break

break	ループ実行を中止します
形式	break

説明

このコマンドは for や foreach や while といったループコマンドの中だけで使用します。break コマンドは TCL_BREAK コードを返すことによって、最も内側のループコマンドに対して即座にリターンすることを通知します。

使用例

```
# st_op.psv から抜粋。操作モードが一致したら foreach のループを break する処理
foreach op $stval(opetype) {
        if { [lindex $op 0] == $ope } {
                set flg 0
                break
        }
        incr openo
}
```

1.5 catch

catch	スクリプトを実行し、例外をトラップします
形式	catch script [varName]

説明

catch コマンドはエラーによってコマンドの解釈実行を中止することを防ぐために使用します。

catch は Tcl インタプリタを再帰的に呼び出して script を実行します。また script 実行中にエラーは評価せずに、常に TCL_OK コードを返します。

catch の返す値は script 実行後に Tcl インタプリタが返すコードを 10 進の文字列にしたものです。返り値は script でエラーが起きなければ値は常に 0(TCL_OK)になります。エラーが起きた場合は例外リターンコードの一つに対応する 0 でない値やエラーメッセージを返します。

varName を指定すると、それは変数名とみし、catch はその変数(結果かエラーメッセージ)をセットします。

使用例

```
set aaa
```

→変数 aaa 未設定時は「can't read "aaa": no such variable」のエラーが発生する。

```
catch {set aaa} bbb
```

→変数$bbb に「can't read "aaa": no such variable」が代入されている。

1.6 cd

cd	作業ディレクトリを変更します
形式	cd [dirName]
説明	
本来の動作としては、カレントディレクトリを dirName に移動し、dirName を指定しない時は、ホームディレクトリ(環境変数 HOME が示すディレクトリ)に移動します。 WebTcl では、物理的なディクトリ移動ができないため、AP サーバ上のコンテンツのパス名間を移動します。 本コマンドは A 版時に作成したスクリプトの互換性のために仕様を調整しています。 相対パス概念のエミュレートでスクリプトの互換性を保つ用途に制限して使用してください。	
使用例	
－	

1.7 close

close	オープンしたファイルをクローズします
形式	close fileId
説明	
fileId で指定するファイルをクローズします。fileId は open コマンドが返した値を指定してください。 このコマンドの後、指定した fileId はファイル操作のコマンドで使用できません。fileId がファイルではなくコマンドパイプラインを指している時には close は子プロセスの終了を待ちます。 通常このコマンドの結果は空文字列ですが、ファイルをクローズする時や子プロセスの終了待ちの時に何か問題がある時は、エラーを返します。	
使用例	
# color.psv から抜粋。ファイルを例外処理込みで close して、問題があれば-1 でリターンする処理 if { [catch { close $fd } str] != 0 } { return -1 } return 0	

1.8 concat

concat	変数リストを結合します。
形式	concat arg [arg ...]

説明

このコマンドは各引数をリストとみなし、それらをつなげて一つのリストにします。 その際、arg の前後の空白を取りさり、また arg と arg の間に区切り用のスペースを一つ挿入します。このコマンドは任意の数の引数を受け付けます。また配列の入れ子になった構造は 1 レベルだけ展開します。

たとえば

concat a b {c d e} {f {g h}}

は

a b c d e f {g h}

と、"{c d e}"と"f"のレベルは展開し、"{g h}"の部分を展開せずに返します。

もし arg を指定しない場合には空文字列を返します。

使用例

st_op.psv から抜粋。新規 stid を全体の管理変数につなげて再代入する処理。

```
set stval(stids) [concat $stval(stids) $stid]
```

1.9 continue

continue	ループコマンドの次の繰り返しまでスキップします
形式	continue

説明

このコマンドは for や foreach や while といったループコマンドの中だけで使用します。

このコマンドは TCL_CONTINUE コードを返すことによって、最も内側のループに対して、ループ本体の残りの実行をスキップし次の繰り返しの実行に移ることを通知します。

使用例

```
# st_sub.psv から抜粋。オブジェクトのコピー時にタイプ不正時に foreach コマンドへループ継続を通知する例。
    foreach i $zid {
…
        switch $zutp {
        line {
…
        default {
st_err <ST_ERROR> タイプ不正 ($zutp)
                continue
            }
```

1.10 eof

eof	オープンしたファイルが EOF に達したか調べます
形式	eof fileId

説明

fileId が EOF 状態になっていれば 1、そうでなければ 0 を返します。fileId は open コマンドが返した値か、標準入出力チャネルを示す stdin、 stdout、 stderr のいずれかでなければなりません。

使用例

```
# color.psv から抜粋。色定義のスクリプトを最終行までループする条件で使用した例。
  proc col_read { filename } {
    variable pb;

    if { [catch { set fd [open $filename r] } str] != 0 } { return -1 }
    if { [catch { gets $fd macro } str] != 0 } { return -1 }
    while { ![eof $fd] } {
      if { [catch { gets $fd macro } str] != 0 } {
        close $fd
        return -1
      }
      scan $macro "%s %s %s %s" dm1 dm2 no rgb
      if { [lsearch -regexp $dm1 "#"] != -1 } { continue }
      if { $no > 0 && $no <= 255 } {
        catch { col_set_pbcolor $pb($no) $rgb } str
      }
    }
    if { [catch { close $fd } str ] != 0 } { return -1 }
    return 0
  }
```

1.11 error

error	エラーの処理を実装します
形式	error message [info] [code]
説明	

TCL_ERROR コードを返すことによって、コマンドの解釈実行を終了させます。Message はアプリケーションに返す文字列で、エラーの原因をアプリケーションに示すためのものです。

info を指定して、空ではなかった場合には、グローバル変数 errorInfo を初期化するのに使用します。

errorInfo はエラーが起きた時のスタックトレース情報を格納する変数です。下位の実行スタックのコマンドからエラーが戻ってきた時に Tcl インタプリタは errorInfo に情報を追加し、スタックしていきます。もし info 引数があれば、それは errorInfo を初期化するのに使用し、さらに Tcl インタプリタがスタックトレース情報を追加するのを最初の一回だけ抑制します。

つまり info を指定した場合、error コマンドを起動したコマンドは errorInfo のスタックには現れないことになります。コマンドが表示される部分には info が入ります。

この機能は catch コマンドと組み合わせると高度なエラーハンドリングが可能です。もしキャッチしたエラーをうまく扱えなかった場合、info を使ってエラーが最初に起こった所を反映したスタックトレースを返すことができます。

※使用例参照

code 引数が指定した時、その値をグローバル変数 errorCode に代入します。

もし code 引数を指定しない時には errorCode は Tcl インタプリタによってエラー処理の一部として自動的に ``NONE'' に設定します。

使用例

```
catch {...} errMsg
set savedInfo $errorInfo
error $errMsg $savedInfo
```

1.12 eval

eval	Tcl スクリプトを評価実行します
形式	eval arg [arg ...]

説明

eval は全体として Tcl のコマンドが入ったスクリプトを構成するような一つ以上の引数を取ります。

eval は concat コマンドと同様に全ての引数を評価して、文字列としてまとめて、その文字列を再帰的に Tcl インタプリタに送り、その実行結果(またはエラー)を返します。

使用例

```
# st_grid.psv から抜粋。変数$stval(proc,stidchk)に格納した文字列を、コマンドとして実行する例。
proc st_set_grid  { stid gc gwin minx miny maxx maxy gw gh } {
          global stval

st_prt <st_set_grid> stid:$stid minx:$minx miny:$miny maxx:$maxx maxy:$maxy gw:$gw gh:$gh

#標準部品ＩＤチェック
          eval $stval(proc,stidchk)
```

1.13 exec

exec	サブプロセスを起動します
形式	exec [switches] arg [arg ...]

説明

このコマンドは引数 arg を一つ以上のサブプロセスの指定と判断します。このコマンドは標準的なシェルのパイプラインのフォーマットを受け付けます。arg のそれぞれがコマンドを構成するワードになり、それぞれのコマンドがサブプロセスになります。

もし exec の第一引数が - で始まっていればそれは exec コマンドに対するオプションスイッチだとみなし、パイプラインとして扱いません。現在次のようなオプションがサポートしています。

-keepnewline

パイプライン出力の最後の改行コードを保存します。指定しない場合最後の改行コードは除去します。

--

オプションの終りを示します。これ以降の引数はたとえそれが - で始まっていたとしても arg であるとみなします。arg（または arg の組）が後述のような形式である場合には、 exec はそれをサブプロセスの入出力フローを制御するのに使用します。これらの引数はサブプロセスには渡しません。

引数に渡さない形式を以下に示します。

|

パイプライン中の個々のコマンドを区切ります。前のコマンドの標準出力と次のコマンドの標準入力をパイプでつなぎます。

|&

パイプライン中の個々のコマンドを区切ります。前のコマンドの標準出力と標準エラー出力の両方を次のコマンドの標準入力にパイプでつなぎます。 この形式のリダイレクションは 2> や >& の指定に優先します。

< fileName

fileName で指定するファイルをオープンして、パイプライン最初のコマンドの標準入力として使います。

<@ fileId

fileId は open コマンドのリターン値など、オープンしたファイルの識別子を指定してください。

これをパイプライン最初のコマンドの標準入力として使用します。 fileId は読み込みモードでオープンしてください。

<< value

最初のコマンドに value が標準入力から渡します。

> fileName

最後のコマンドの標準出力が fileName というファイルにリダイレクトします。ファイルの内容は上書きします。

説明

2> filename

パイプライン全コマンドの標準エラー出力を fileName というファイルにリダイレクトします。ファイルの内容は上書きします。

>& fileName

最後のコマンドの標準出力と全てのコマンドの標準エラー出力を fileName というファイルにリダイレクトします。ファイルの内容は上書きします。

>> fileName

最後のコマンドの標準出力を fileName というファイルにリダイレクトします。ファイルの最後に追加書き込みします。

2>> fileName

パイプライン全コマンドの標準エラー出力を fileName というファイルにリダイレクトします。ファイルの最後に追加書き込みします。

>>& fileName

最後のコマンドの標準出力と全コマンドの標準エラー出力を fileName というファイルにリダイレクトします。ファイルの最後に追加書き込みします。

>@ fileId

fileId は open コマンドのリターン値など、オープンしたファイルの識別子を指定してください。最後のコマンドの標準出力を fileId のファイルにリダイレクトします。fileId は書き込みモードでオープンしてください。

2>@ fileId

fileId は open コマンドのリターン値など、オープンしたファイルの識別子を指定してください。すべてのコマンドの標準エラー出力を fileId のファイルにリダイレクトします。fileId は書き込みモードでオープンしてください。

>&@ fileId

fileId は open コマンドのリターン値など、オープンしたファイルの識別子でなければなりません。最後のコマンドの標準出力とすべてのコマンドの標準エラー出力を fileId のファイルにリダイレクトします。fileId は書き込みモードでオープンしてください。標準出力をリダイレクトしない場合には exec はパイプライン最後のコマンドの標準出力内容を返します。パイプラインのコマンドのどれかが異常終了したり kill されたりサスペンドさせられたた場合には exec はエラーを返し、エラーメッセージにはパイプラインの出力とそれに続いて異常終了を知らせるメッセージが入ります。また、 errorCode 変数には最後の異常終了に関する情報が入ります。もしコマンドのどれかが標準エラー出力に書き出し、標準エラー出力がリダイレクトされていなかった場合には exec はエラーを返し、エラーメッセージにはパイプラインの出力に続いて（もしあれば）異常終了についてのメッセージ、さらに標準エラーの出力が入ります。

説明

結果やエラーメッセージの最後の文字が改行コードだった場合には、その改行コードを削除します。これは他の Tcl のリターン値が改行で終了しないので、それに合わせるためです。しかし-keepnewline オプションを指定していれば最後の改行コードを保存します。

標準入力を``<'' や ``<<'' や ``<@'' でリダイレクトしない時にはパイプライン最初のコマンドの標準入力はアプリケーションの標準入力になります。

最後の引数が ``&'' であった場合には、そのパイプラインはバックグラウンドで実行されます。この場合 exec コマンドはパイプラインの全てのサブプロセスのプロセス識別子からなるリストを返します。パイプラインの最後のコマンドの標準出力は、リダイレクトされていなかった場合アプリケーションの標準出力に出ます。またパイプラインの全てのコマンドのエラー出力はリダイレクトされていない場合アプリケーションの標準エラー出力に出ます。

各コマンドの最初のワードはコマンド名であるとみなされます。``~'' の拡張が行なわれ、その結果にスラッシュが入っていなければ環境変数 PATH を使って実行ファイルが探されます。スラッシュが入っていればそれはカレントディレクトリから到達可能な実行可能ファイルを指していなければなりません。コマンドの引数に対して ``glob'' 展開やその他のシェルで行われるような置き換えは行われません。

使用例

```
# メモ帳をフォアグランドで実行する場合。メモ帳終了まで次の Tcl コマンドが実行できない
exec notepad
# メモ帳をバックグラウンドで実行する場合。メモ帳終了せずに次の Tcl コマンドが実行可能
exec notepad &
```

1.14 exit

exit	プロセスを終了します
形式	exit [returnCode]

説明

プロセスを終了させ、returnCode(整数)を終了コードとしてシステムに返します。returnCode を指定しない場合は、省略時の値は 0 です。

プロセスとは WebTcl エンジン部分を示します。WebTcl エンジンは終了時に FE-COM にも終了を通知して、WebTcl 全体を終了します。

使用例

```
# WTCLSampMain.psv から抜粋。各種終了処理の最後に exit でプロセス終了を実行し、WebTcl 全体を終了します。
proc ExitProg { } {
  global map
  #各種クローズ
  st_close           $map(st)
  g_close_layer $map(gc)
  g_close_dlist $map(gc)
  g_close_gwin $map(gc)
  g_close_context $map(gc)
  w_destroy $map(top1)
  unset map
  namespace delete WTCD
  namespace delete WTPD
  exit
}
```

1.15 expr

expr	式を評価します
形式	expr arg [arg arg ...]

説明

arg を一つにまとめ(ただし各引数の間に区切りの空白を挿入します)その結果を Tcl の式として評価し、その値を返します。Tcl で使える演算子は C 言語の の演算子のサブセットで、対応する C 言語 の演算子と同じ意味と優先度を持っています。式はほとんどの場合結果として数を返します（整数または実数)。

たとえば次の式

expr 8.2 + 6

を評価すると “14.2”が結果として返ってきます。Tcl の式が C 言語の式と異なるのはオペランドの指定の仕方です。また Tcl では数でないオペランドや文字列比較をサポートしている点も異なります。

以下 expr の引数で指定する、①オペランド、②演算子、③数学関数、④データ型、オーバーフロー、精度、⑤文字列操作に関して項目別に説明します。

①オペランド

Tcl の式はオペランドと演算子とかっこの組合せです。オペランド、演算子、かっこのあいだに空白文字を入れても、Tcl の演算機能はそれを無視します。オペランドは可能なかぎり整数であると解釈します。

整数は 10 進(通常のケース)でも、8 進(0 ではじまっている場合)でも、 16 進(0x ではじまっている場合)でも指定することができます。もしもオペランドが上に挙げた形式ではなかった場合、実数として解釈を試みます。

実数は ANSI 対応の C 言語 コンパイラが受け付けるどんなフォーマットででも指定できます（ただし``f''、``F''、 ``l''、``L'' といったサフィックスは使えないケースがあります。）。

たとえば、2.1、3.、6e4、7.91e+16 はそれぞれ正しく実数と解釈します。もしもオペランドを数として解釈することができなければ、単なる文字列だと判定します（そして文字列操作しか適用できません。）。

オペランドは次のどの方法でも指定できます。

[1] 整数値か実数値として。

[2] 標準の $ 記法を使った Tcl の変数として。

変数の値をオペランドとして使います。

[3] ダブルクォートで囲んだ文字列として。

式の解釈時にクォートで囲んだ部分に対してバックスラッシュ、変数、コマンドの各置換を行ない、その結果をオペランドとして使用します。

[4] 中かっこで囲んだ文字列として。

開き中かっこと対応する閉じかっこで囲んだ部分を一切置き換えされることなくそのままオペランドとして使います。

[5] 大かっこで囲んだ Tcl コマンドとして。

そのコマンドを実行し、結果の文字列をオペランドとして使います。

説明

[6] 数学関数として。

関数の引数には前述のどの形式のオペランドも使えます。例えば ``sin($x)'' など。使用できる関数の詳細は２章の「数学関数」を参照してください。。

上に述べた置換（たとえばクォートした文字列の中）は Tcl の演算機能によって行なわれます。しかし演算機能を呼出す前に、すでにコマンドの解釈によって置換えが行なわれていることがありえます。

不要な置換えを避けるため、式は中かっこで括り、コマンド解釈による置き換えを避けてください。

たとえば変数 a の値が 3 で、 b の値が 6 だとします。このとき次の各行の左にある式を評価すると右にある値になります。

```
expr 3.1 + $a 6.1
expr 2 + "$a.$b" 5.6
expr 4*[llength "6 2"] 8
expr {{word one} < "word $a"}0
```

②演算子

使用可能な演算子の一覧を、優先度の高い方から順にグループ化して示します。

- ~ !

単項演算子のマイナス、ビット演算子の NOT、論理演算子の NOT。これらは文字列のオペランドには適用できません。またビット演算子の NOT は整数以外には適用できません。

* / %

かけ算、割算、剰余。これらは文字列のオペランドには適用できません。また剰余演算子は整数以外には適用できません。剰余の値は常に除数と同じ符号を持ち、その絶対値は除数より小さくなります。

+ -

足し算と引き算。数値であればどんなオペランドにも適用できます。

<< >>

左シフトと右シフト。整数にしか適用できません。

< > <= >=

比較演算子: 小なり、大なり、以下、以上。それぞれ条件が正しければ 1、そうでなければ 0 を返します。これらの演算子は数値だけではなく文字列に対しても適用でき、その場合には文字列比較が行なわれます。

== !=

比較演算子: 等値と非等値。それぞれ 1 か 0 を返します。どんなオペランドに対しても適用できます。

&

ビット演算子の AND。整数にしか適用できません。

^

ビット演算子の exclusive OR。整数にしか適用できません。

説明

|

ビット演算子の OR。整数にしか適用できません。

&&

論理演算子の AND。オペランドが両方とも 0 でないならば 1、そうでなければ 0 を返します。数値のオペランド（整数または実数)にしか適用できません。

||

論理演算子の OR。オペランドが両方とも 0 ならば 0、そうでなければ 1 を返します。数値のオペランド（整数または実数）にしか適用できません。

x?y:z

C 言語の 3 項演算子と同じです。もし x を評価した値が 0 でないならば結果は y の値、そうでないならば結果は z の値になります。 x オペランドは数値しか使用できません。

同じ優先順位の 2 項演算子は左から右の順で評価されます。

たとえば

expr 4*2 < 7

は 0 になります。

&&、|| と ?: の各演算子は C と同じく「遅延評価」を行います。つまりオペランドは結果を計算するのに必要でないかぎり評価しない、ということです。

たとえば

expr {$v ? [a] : [b]}

では $v の値によって [a] と [b] のどちらか一方しか実際には評価しません。しかし、実際にはこれは式全体を中かっこでくくった場合だけです。くくらなかった場合には expr を実行する前に Tcl のパーサが [a] と [b] の両方を実行します。

③数学関数

Tcl では次のような数学関数を式の中で使うことができます。

acos cos hypot sinh asin cosh log sqrt atan exp log10 tan atan2 floor pow tanh ceil fmod sin

これらの関数は数学ライブラリに入っている同名の関数を呼出します。個々の関数の動作については２章の「数学関数」を参照してください。 Tcl ではさらに次のような整数と実数の変換のための関数を使うことができます。

abs(arg)

arg の絶対値を返します。 Arg は整数でも実数でもよく、結果は引数と同じタイプになります。

double(arg)

arg が実数なら arg をそのまま返します。そうでなければ arg を実数に変換したものを返します。

int(arg)

arg が整数なら arg をそのまま返します。そうでなければ arg を小数以下を切り捨てて整数に変換したものを返します。

説明

round(arg)

arg が整数なら arg をそのまま返します。整数以外の場合、arg の小数以下を四捨五入して整数に変換したものを返します。

④データ型, オーバーフロー、精度

内部での演算は整数の場合 C 言語 の long 型で、実数の場合は C 言語の double 型で行なわれます。文字列を実数に変換する際の指数部のオーバーフローはチェックして、オーバーフローと判定した時は Tcl エラーとします。文字列から整数への変換についてはオーバーフローの検出は C 言語ランタイムの動作に依存します。

実数のオーバーフローやアンダーフローに関してはハードウェアでサポートしている範囲で検出します。

整数、実数、文字列オペランド間の内部表現の変換は必要に応じて自動的に行なわれます。数値計算に関しては、実数が出てくるまでは整数で行なわれ、出てくるとそのあとは実数で行なわれます。

たとえば、

expr 5 / 4

は 1 を返しますが

expr 5 / 4.0

expr 5 / ([string length "abcd"] + 0.0)

はともに 1.25 を返します。実数は整数と区別できるよう常に小数点または ``e'' 付きで返します。

例えば

expr 20.0/5.0

は ``4'' ではなく ``4.0'' を返します。グローバル変数 tcl_precision で実数値を文字列に変換する時の有効桁数を指定することができます（ただし最後のゼロは除きます)。

tcl_precision を設定していない時は、デフォルトの 6 桁の精度になります。実数部の全ての有効桁数を有効にしたい場合は、tcl_precision を 17 に設定してください。有効桁数が 17 桁の場合、値を一度 17 桁の文字列に変換して、後で計算のためにバイナリ表現に戻した時、そのバイナリ表現は元と同じであることを保証します。

⑤文字列操作

比較演算子のオペランドとして文字列を使うことができますが、式の評価の際には可能ならば整数または実数として比較がなされます。もしも比較のオペランドの一方が文字列でもう一方が数値の場合には数値は C 言語ランタイムの sprintf を内部的に用いて、整数の場合 %d、実数の場合には %g で文字列に変換してから比較が行なわれます。

たとえば、次の式

expr {"0x03" > "2"}

expr {"0y" < "0x12"}

の値は両方とも 1 になります。最初の比較は整数値の比較で行なわれ、2 番目のは 2 番目のオペランドを文字列 ``18'' に変換してから文字列比較がなされます。

使用例

```
# wsamp020.psv から抜粋。ループで生成するラベルの Y 座標を計算しています。
for {set i 0} {$i < 9} {incr i} {
          w_label ws020(lb$i) $ws020(top) -x 420 -y [expr 10 + $i*25]¥
                    -w 100 -h 20 -f $ws020(fg) -b $ws020(bg2) -l [lindex $ws020(label) $i]¥
                    -v m7 -a c
          set ws020(tx$i) [w_text_box ws0202(tx$i) $ws020(top)¥
                    -x 520 -y [expr 10+$i*25] -w 10 -h 1 -f $ws020(fg) -b $ws020(bg)¥
                    -v m7 -m s -l 8192 -s on -c ″
                              w_set_value    $ws020(win)    [lindex    $ws020(opt)    $i]    ¥[w_text_str
¥$ws020(tx$i)¥]
                    ″ ]
}
```

1.16 file

file	ファイルの名前や属性の操作を行います
形式	file option name [arg arg ...]

説明

このコマンドはファイルの名前や属性に関する操作を提供します。

name はファイルの名前を指定します。``~'' ではじまっていればそのコマンドを実行する前に展開します。

option でファイルに対する操作を指定します。指定できるオプションは次の通りです。

file atime name

name で指定するファイルの最終アクセス時間を示す 10 進の数値の文字列を返します。時間は特定の日時（1970 年 1 月 1 日）からの秒数です。ファイルが存在しない時、アクセス時間を調べることができなかった時にはエラーを返します。

file dirname name

最後の要素を除いて name のパスの全てのコンポーネントから成る名前を返します。もし name が相対的なファイル名であり、ただ 1 つのパス要素を含む場合、"."を返します。name がルートディレクトリを参照する場合、ルートディレクトリが返されます。

file executable name

name がユーザによって実行可能なら 1、そうでないなら 0 を返します。

file exists name

name が存在し、途中のディレクトリの移動のアクセス権限があれば 1、そうでなければ 0 を返します。

file extension name

name の最後のドットからあと（そのドットも含む）の文字列を返します。 name にドットが無い場合、空文字列を返します。

file isdirectory name

ファイル name がディレクトリなら 1、そうでなければ 0 を返します。

file isfile name

ファイル name が通常のファイルなら 1、そうでなければ 0 を返します。

file lstat name varName

stat() の代わりに lstat() システムコールを使うことを除けば stat(後述) と同じです。これは name がシンボリックリンクだった場合に、 varName に返す情報がそのリンクがさしているファイルのものではなく、リンクそのものについてのものであるということです。

file mtime name

ファイル name の最終修正時間を示す 10 進の数値の文字列を返します。時間はある特定の日時（1970 年 1 月 1 日）からの秒数です。ファイルが存在しない時、修正時間を調べることができなかった時にはエラーを返します。

説明

file owned name

ファイル name がユーザのものであれば 1、そうでなければ 0 を返します。

file readable name

ファイル name をユーザが読むことができれば 1、そうでなければ 0 を返します。

file readlink name

name で指定するシンボリックリンクの値（つまりそれが指しているファイルの名前）を返します。 name がシンボリックリンクでない時、あるいは値を読むことができない時はエラーを返します。

file rootname name

name の最後のドットまで(ただしそのドットは含まない)の文字列を返します。name にドットを含んでいない時は name が返ります。

file size name

ファイル name のサイズ（単位はバイト）を示す 10 進の数値の文字列を返します。ファイルが存在しない時、サイズを調べることができなかった時にはエラーを返します。

file stat name varName

name に対して stat() システムコールを呼びだし、その結果を varName で指定する変数に代入します。 varName は配列変数として扱い、次のような要素をセットします

atime, ctime, dev, gid, ino, mode, mtime, nlink, size,type, uid. type

これ以外の要素はそれぞれ stat() の返す構造体の対応する数値を 10 進で表した文字列です。それぞれの値の意味については C 言語ランタイムの stat() のマニュアルを参照してください。type にはファイルのタイプがコマンド file type と同じ形式で入ります。このコマンドは空文字列を返します。

file tail name

name の最後の ``/'' からあとの文字列を返します。 name が ``/'' を含んでいなければ name を返します。

file type name

ファイル name のタイプを示す文字列を返します。タイプは

file、 directory、 characterSpecial、 blockSpecial、fifo、 link、 socket

のいずれかです。

file writable name

ファイル name がユーザによって書き込み可能なら 1、そうでなければ 0 を返します。

使用例

```
# color.psv から抜粋。$c(file)のファイル名からパス名と、ファイル名を取り出しています。
          set path     [file dirname $c(file)]
          set filename [file tail $c(file)]
```

1.17 flush

flush	バッファリングしている出力をフラッシュします
形式	flush fileId

説明

fileId に関して、バッファしているデータを出力にフラッシュします。fileId は open コマンドが返した値か、あるいは標準入出力チャネルを示す stdin、stdout、stderr のいずれかを指定してください。このコマンドは空文字列を返します。

本機能は put 直後に、出力に書き込みを確実に行いたい場合使用してください。ただし書き込みを確実に行うと、出力時のバッファ機能が無効になり、処理性能自体は低下しますので、適切な個所で使用してください。

使用例

```
#デバッグ用のファイル書き込み情報を、put で出力直後にフラッシュしている例。
set fd [open "c:/temp/debug.log" "a+"]
…
puts $fd "DEBUG:${debugInformation}"
flush $fd
…
close $fd
```

1.18 for

for	For ループコマンド
形式	for start test next body

説明

for は C 言語の for 文と同じような構造を持つループ制御コマンドです。

引数 start、 next および body は Tcl コマンド文字列を指定します。test は式の文字列です。

for コマンドは、まず start を実行します。その後 test を式として評価を行います。

評価の結果が 0 でなければ body を実行し、その後 next を実行して、そしてループを繰り返します。 この繰り返しは test を評価した値が 0 になった時に終了します。

body の中で continue コマンドが実行されるとその時の body の残りのコマンドの実行をスキップします。

もし body や next の中で break コマンドが実行されると for コマンドは直ちにループを終了します。

break と continue の動作は C 言語の仕様と同様です。

for は空文字列を返します。

使用例

```
#charlist.psv から抜粋。２バイトの文字コードのリストを生成するため、
#１バイト目と２バイト目のコード生成でループしています。
for {set i $ssb} {$i <= $seb} {incr i } {
        set strlist ""
        for {set j 0} {$j <= 15} {incr j} {
                set lmb [expr $i * 16 + $j]
…
                append strlist $val " " [expr 0x81] " " [expr 0x40] " "
        }
        set binlist [binary format c* $strlist]

        append chlt(obj_list) " " [g_text $chlt(gct) 10 [expr 10 + (26 - $chlt(obj_cnt)) * 60] 32 $binlist
-s 0 -f $chlt(ftype)]
        set chlt(obj_cnt) [expr $chlt(obj_cnt) + 1]
}
```

1.19 foreach

foreach	リストのすべての要素について繰り返し処理を行います
形式	foreach varname list body

説明

このコマンドは、 varname は変数名、 list は varname に代入する値のリスト、body は Tcl コマンド文字列を指定します。list の各フィールドについて、 foreach は lindex コマンドと同じく、左から右の順で、そのフィールドの内容を varname に代入します。そしてフィールドごとに body を実行します。

body の中で break や continue を使用可能です。これらは for コマンドの時と同じ動作を行います。

foreach は空文字列を返します。

使用例

```
#st_op.psv から抜粋。配列 stval の要素でループし、
#クローズする$stid と同じ要素があった場合、変数を unset しています。
proc st_close { stid } {
    global stval
…
    foreach i [lsort [array name stval]] {
        if { [regexp $stid $i] } {
            unset stval($i)
        }
    }
}
```

1.20 format

format	sprintf スタイルで文字列をフォーマットします
形式	format formatString [arg arg ...]

説明

このコマンドは内部的に C 言語ランタイムの sprintf を使い、フォーマットした文字列を作ります。formatString で結果をどのようにフォーマットするかの、書式を指定します。sprintf のように%フィールドを使用し、必要ならばその他の引数で結果と置き換えるための値を指定します。format はフォーマットした文字列を返します。

①フォーマット詳細

このコマンドは formatString を左から右にスキャンしながらフォーマットを実行します。フォーマット文字列の文字は、%フィールド以外の文字列はそのまま結果の文字列に追加します。%フィールドは直接結果の文字列にコピーしません。%フィールドの後の文字が変換の指定子です。この変換指定子にしたがって次の arg を変換して、変換した結果を、フォーマット後の文字列に追加します。フォーマット文字列の中に複数の変換指定子があれば、それぞれを arg に対応付けて変換します。 format コマンドには formatString 中の変換指定子に合うだけの数の arg を指定しなければなりません。

各変換指定子は以下の６つの要素です。

・XPG3 で規定する位置指定子

・フラグ

・フィールドの最小幅

・フィールドの精度

・フィールドの長さ修飾子

・変換指定文字

変換指定文字以外は全て省略可能です。フィールドを指定する場合には上にあげた順番通りに指定する必要があります。以下、各フィールドについて順に説明します。

(a) %の後に数字が続きそのあとに $ がきた場合、例えば``%2$d'' のような場合には、変換が行われる引数は通常のようにまだ使われずに残っている引数の最初のものとはなりません。
そうではなく、番号で指定した引数を使います。例えば 1 ならば arg の先頭のものを使用します。もしも変換指定が複数の引数を必要とする時、つまり指定子の中に * 文字が入っている時には番号で指定する引数から連続する引数が順に使います。これは XPG3 の位置指定の方式に従ったものです。
formatString の中に位置指定が一つでもあれば､全ての変換指定子に位置指定を含む必要があります。

(b) 変換指定子の 2 番目のフィールドには次のフラグ用文字を任意の数、任意の順で書くことができます。

-
変換対象の引数を左寄せにすることを指定します（数字は通常必要ならばスペース文字を入れて右寄せになります）。

説明

+

正の数字でも常に符号付きでフォーマットします。

space

数字をプリントする場合、先頭の文字が符号でなければスペースを追加することを指定します。

0

数字の先頭のパディングにスペースではなく 0 を使うことを指定します。

#

別形式での出力を指定します。 o および 0 変換では最初の文字が常に 0 になります。 x および X 変換では、結果が 0 以外の時にはそれぞれ 0x および 0X が先頭に追加します。全ての実数変換（e、E、 f、g、 G）では結果に常に小数点がつくことが保証します。 g および G 変換では最後の 0 を省略しません。

(c) 変換指定子の 3 番目のフィールドはその変換の最小フィールド幅を指定する数字です。

これは通常表形式の出力で各欄の位置を揃えるのに使用します。変換した結果の文字数がフィールドの最小幅より少なければ最小幅になるまでパディングします。通常パディングは変換した結果の左側に余分のスペースを入れることで行われますが、0 や-フラグによりそれぞれ 0 でパディングしたり右側にスペースを入れたりすることも可能です。最小幅が数字ではなく * で指定した場合には format コマンドの次の引数が最小幅指定とみなします。これは数字でなくてはなりません。

(d) 変換指定子の 4 番目のフィールドは精度を表すもので、ピリオドとそれに続く数字で指定します。

数字の意味は変換によって異なります。 e、 E、 f 変換においては小数点の右側の数字の桁数の指定になります。 g と G 変換では小数点の両側の桁数を合わせたトータルの桁数の指定になります(ただし小数部分の最後の 0 は #フラグが指定されないかぎり取り去られます)。 整数の変換においては出力するする数字の最小の桁数の指定になります(必要ならば先頭に 0 が追加されます)。s 変換ではプリントする最大の文字数の指定になります。文字列がこれより長い場合、残りの文字は捨てられます。精度が数字ではなく * で指定された場合には format コマンドの次の引数が精度指定とみなされます。これは数字でなくてはなりません。

(e) 変換指定子の 5 番目のフィールドは長さの修飾子で h か l でなければなりません。

h なら数字は変換の前に 16 ビットに切り捨てられますが、これはほとんど使い道がありません。l 修飾子は無視されます。

(f) 変換指定子の最後のフィールドはどの種類の変換をするのかを指定するアルファベットです。

現在、次のような変換文字を使うことができます。

d

整数を符号つき 10 進文字列に変換します。

u

整数を符号なし 10 進文字列に変換します。

説明

i

整数を符号つき 10 進文字列に変換します。整数は 10 進でも 8 進(先頭が 0)でも 16 進(先頭が 0x)でも構いません。

o

整数を符号なし 8 進数に変換します。

x or X

整数を符号なし 16 進数に変換します。数字として x の時には ``0123456789abcdef'' が、X の時には``0123456789ABCDEF'' を使用します。

c

整数を 8 ビットキャラクタに変換します。

s

変換は行われません。単に文字列を挿入します。

f

実数を xx.yyy という形式の符号付きの 10 進文字列に変換します。y の個数は精度指定によって決まります(デフォルトは 6)。精度指定が 0 の場合には小数点は出力しません。

e or E

実数を x.yyye_zz という数学記法に変換します。y の個数は精度指定によって決まります(デフォルトは 6)。精度指定が 0 の場合には小数点は出力しません。 E 変換の場合、e の代わりに E を出力します

g or G

もし指数部が-4 より小さいか、または精度指定以上であれば実数を %e や %E 形式で変換します。それ以外の時は%f を適用します。最後の 0 および小数点は出力しません。

%

変換は行われません。単に% を挿入します。

数値変換の場合、変換が行われる引数は整数または実数を示す文字列でなくてはなりません。 format は引数を一旦バイナリ表現に直し、それを変換指定子にしたがって再び文字列に変換します。

②C 言語の sprintf 関数との違い

format コマンドの動作は C 言語の sprintf 関数とほぼ同じですが、次の点が異なります。

[1] %p と %n 変換をサポートしていません。

[2] %c 変換では引数は 10 進の文字列でなければなりません。これを対応する文字に変換します。

[3] l 修飾子は無視します。整数値は常に修飾子なしの状態で変換し、実数値は常に l 修飾子がついているかのような状態で変換します（内部表現としては double を使用）。

h 修飾子が指定された時には、整数値は変換前に short に直されます。

使用例

```
#color.psv から抜粋。
proc col_sl_label {red green blue} {
        set r [format "%2.2x" $red]
        set g [format "%2.2x" $green]
        set b [format "%2.2x" $blue]
        w_set_value @col_lb4 -l $red
        w_set_value @col_lb5 -l $green
        w_set_value @col_lb6 -l $blue
        w_set_value @col_lb3 -l "&$r$g$b"
        w_set_value @col_lb8 -b "&$r$g$b"
        if { [expr ($red + $green + $blue) / 3 - 0x80] > 0 } {
                w_set_value @col_lb8 -f &000000
        } else {
                w_set_value @col_lb8 -f &ffffff
        }
}
```

1.21 gets

gets	ファイルから一行読みます
形式	gets fileId [varName]

説明

このコマンドは fileId で指定するファイルから一行読み、最後の改行コードを除去します。varName を指定した時は、その変数に読んだ行を入れ、その文字数(改行は含まない)を返します。もし、1 文字も読んでいないのに EOF になった時には -1 を返し、 varName には空文字列が入ります。 varName の指定を省略した時は、その行そのもの(改行は除く)を返します。1 文字も読んでいないのに EOF になった時には空文字列を返します。行が改行以外に文字を含んでいなかった時も空文字列が返るので、ファイル末端の検出には eof を使う必要があります。ファイルの最後の文字が改行コードではなかった場合、 gets はファイルの最後に改行コードがある時と同じ動作します。 fileId は stdin か、open の返した値を指定してください。対象のファイルは、読み込み用にオープンしたファイルを指定してください。gets コマンドは実行前にファイル関連の EOF やエラー状態のフラグをクリアします。

使用例

#color.psv から抜粋。色定義ファイルを行単位で読み込みます。

```
proc col_read { col filename } {
	upvar $col c
	if { [catch { set fd [open $filename r] } str] != 0 } { return -1 }
	if { [catch { gets $fd macro } str] != 0 } { return -1 }
	while { ![eof $fd] } {
		if { [catch { gets $fd macro } str] != 0 } {
			close $fd
			return -1
		}
		scan $macro "%s %s %s %s" dm1 dm2 no rgb
		if { [lsearch -regexp $dm1 "#"] != -1 } { continue }
		if { $no > 0 && $no <= 255 } {
			catch { col_set_pbcolor $c(pb$no) $rgb } str
		}
	}
	if { [catch { close $fd } str ] != 0 } { return -1 }
	return 0
}
```

1.22 glob

glob	パターンにマッチするファイル名を返します
形式	glob [switches] pattern [pattern ...]

説明

このコマンドはファイル名の展開(``globbing'')を行います。引数 pattern にマッチするファイル名のリストを返します。glob の最初の引数が - で始まっているとそれはオプションスイッチであると判定します。

使用可能なオプションを以下に示します。

-nocomplain

マッチするものが一つもなかった時、空のリストを返すようにします。このオプションを指定しない場合、マッチしなかった時にはエラーを返します。

--

オプションの終りを示します。これ以降の引数はたとえそれが-で始まっていたとしても pattern であると判定します。

pattern には次のような特殊文字を使うことができます。

?

任意の 1 文字にマッチします。

*

任意の 0 文字以上の文字列にマッチします。

[chars]

chars に含まれる任意の 1 文字にマッチします。 chars に a-b という形式の指定があった場合には a から b まで (ab も含む) の任意の文字にマッチします。

¥x

文字 x にマッチします。

{a,b,...}

リストに含まれる任意の文字列 (a、 b など) にマッチします。``.'' で始まるファイル名や ``/.'' などに対してはそれを明示的に指定するか、 {} を使用しない限りマッチは起こりません。また、 ``/'' はすべて明示的に指定しないとマッチしません。

~

pattern の先頭の文字が ```~'' の時には、```~'' の後の文字列と同じ名前のユーザのホームディレクトリを指します。 ```~'' の直後の文字が ``/'' だった場合には環境変数 HOME の値を適用します。

glob の結果のリストはソートしていません。ソートしたリストが必要ならば lsort コマンドを使用してください。

使用例

```
# フルパスのローカルファイル名から、同一ディレクトリのファイル名のリストを取得しています。
cd $path
set target [file tail $fname]
# 対象ファイルを全件取得
if { [catch { set files [glob -nocomplain $target*] }] } {
...
}
```

1.23 global

global	グローバル変数にアクセスします
形式	global varname [varname ...]

説明

このコマンドは Tcl プロシジャの外では無効です。プロシジャ内であれば varname で指定する変数がローカルではなく、グローバルであることを宣言します。そのプロシジャ内では、varname はローカルではなくグローバル変数として扱います。

使用例

```
# st_op.psv から抜粋。外部変数の管理情報を格納した配列 stval を参照し、内容をリセットしています。
proc st_reset_val { stid } {
        global stval

#標準部品ＩＤチェック
        eval $stval(proc,stidchk)

        set stval($stid,pnum) 0
        set stval($stid,gpoint) ""
        set stval($stid,param) ""
        set stval($stid,anydata) ""
        set stval($stid,olddata) ""
        set stval($stid,zuobjs) ""
        set stval($stid,rubber) 0
}
```

1.24 if

if	スクリプトを条件実行します
形式	if expr1 [then] body1 elseif expr2 [then] body2 elseif ...[else] [bodyN]

説明

if コマンドは expr1 を式として評価※1します。

※1：expr コマンドが引数を評価するのと同じ仕様で評価します。

式の値は真偽値として評価します（数値の場合、0 が偽でそれ以外は真になります。文字列の場合 true や yes が真、 false や no が偽になります）。

評価の結果が真であれば body1 を実行します。評価の結果が偽の場合、expr2 を評価し、評価の結果が真ならば body2 を実行する…という形式で評価式と実行文の組合せを順次チェックしていきます。もしどの式も真でなかった時は else に対応する bodyN を実行します。then と else の指定は省略可能です。これらはスクリプトを読みやすくするためのものです。 elseif 部分はいくつあっても（0 個でも）構いません。bodyN も省略可能ですが、省略時は else を指定してはいけません。if の返す値は実行したコマンドの値です。もしすべての式が偽で bodyN もなければ空文字列を返します。

使用例

st_op.psv から抜粋。オペレーションタイプが一致するか確認しています。

```
#オペレーションタイプチェック
    set openo 0
    set flg 1
    foreach op $stval(opetype) {
        if { [lindex $op 0] == $ope } {
            set flg 0
            break
        }
        incr openo
    }
```

1.25 incr

incr	変数の値を増加します
形式	incr varName [increment]

説明

varName で指定する変数の値を増やします。変数の値は整数値だけ指定可能です。increment を指定すると、その値(これも整数で指定)を変数 varName に加算します。increment を省略した時は 1 を varName に加算します。結果の値は 10 進の文字列として変数 varName に代入し、コマンドの結果としても返します。

使用例

#color.psv から抜粋。ループ時の変数$i の加算を行っています。

```
for {set i 1} {$i <= 255} {incr i} {
        set bg [w_get_value $c(pb$i) -b]
        if { [catch { puts $fd  "¥tg_alloc_color ¥$gc $i $bg" } str] != 0 } {
                return -1
        }
}
```

1.26 info

info	Tcl のインタプリタの状態に関する情報を返します
形式	info option [arg arg ...]

説明

このコマンドは Tcl インタプリタの内部情報を提供します。option により取得できる情報を選択できます。

info args procname

procname で指定するプロシジャの引数の名前を順に並べたリストを返します。 procname は Tcl コマンドプロシジャの名前を指定してください。

info body procname

プロシジャ procname の本体を返します。procname は Tcl コマンドプロシジャの名前を指定してください。

info cmdcount

実行したコマンドの数の合計を返します。

info commands ?pattern?

pattern を省略した時は、組み込みコマンドと proc コマンドで定義されたコマンドプロシジャを含む全ての Tcl コマンドの名前のリストを返します。 pattern を指定した時は、 pattern にマッチする名前だけが返します。マッチングは string match と同じ規則を用います。

info complete command

command で指定する入力が、Tcl コマンドとして完全であれば 1 を返します。コマンドが入力中で Tcl コマンドとして不完全な時は 0 を返します。このコマンドは行ごとの入力を行うような環境で、複数行にわたるコマンドをユーザが入力する時のチェックに使用できます。もしコマンドの入力がまだ途中の時は、入力が終了するまで実行を遅らせる処理がスクリプトで実装できます。

info default procname arg varname

procname は Tcl コマンドプロシジャの名前を指定し、arg はそのプロシジャの引数の名前を指定します。arg がデフォルト値を持たなければこのコマンドは 0 を返します。デフォルト値を持つ場合は 1 を返し、arg のデフォルト値を変数 varname にセットします。

info exists varName

varName という名前の変数がグローバルまたはローカル変数として存在すれば 1、そうでなければ 0 を返します。

info globals ?pattern?

patter の指定省略時は、現在存在する全グローバル変数の名前のリストを返します。 pattern を指定した時は、pattern にマッチする名前だけを返します。マッチングは string match と同じ規則を用います。

説明

info level ?number?

number の指定省略時は、このコマンドはこのコマンドを起動したプロシジャのスタックレベルを返します。コマンドがトップレベルの場合には 0 を返します。number を指定した時はスタックレベル number のプロシジャコールの名前と引数が入ったリストを返します。 number が正の時は、特定のスタックレベル（1 は一番上位のアクティブなプロシジャ、2 はそれに呼出したプロシジャ、など）を指します。number が負の時は、現在のレベルからの相対値（0 はカレントプロシジャを指し、-1 はそれを呼出したプロシジャ、など）と判断します。スタックレベルの意味については uplevel コマンドを参照してください。

info library

Tcl の標準スクリプトが入っているライブラリディレクトリの名前を返します。

PreSerV ではライブラリを使用しないため、常にエラーになります。

info locals ?pattern?

pattern の指定省略時は、現在存在する全ローカル変数の名前と、現在のプロシジャの引数も含めて、リストにして返します。globa や upvar コマンドで定義した変数は返しません。 pattern を指定した時は、pattern にマッチする名前だけを返します。マッチングは string match と同じ規則を用います。

info patchlevel

Tcl の現在のパッチレベルを示す 10 進の整数を返します。パッチレベルは新しいリリースやパッチのたびに値を増加し、これによって Tcl のオフィシャルバージョンを識別することができます。

PreSerV では、オフィシャルバージョンに追従しないため、常に"8.0.5"を返します。

info procs ?pattern?

pattern の指定省略時は、全 Tcl コマンドプロシジャの名前のリストを返します。 pattern を指定した時は、pattern にマッチする名前だけを返します。マッチングは string match と同じ規則を用います。

info script

Tcl のスクリプトファイルが現在実行中の時は、実行中の最も内側のファイル名を返します。実行中のスクリプトファイルが無い場合は、空文字列を返します。

info tclversion

この Tcl のバージョン番号を x.y という形式で返します。

PreSerV は Tcl のバージョンアップは行わないため、常に"8.0"を返します。

info vars [pattern]

pattern の指定省略時は、ローカル変数や現在アクセスできるグローバル変数を含めて、現在アクセスできる全変数の名前のリストを返します。 pattern を指定した時は、pattern にマッチする名前だけを返します。マッチングは string match と同じ規則を用います。

使用例

#st_op.psv から抜粋。$stval($stid,grid_proc)にグリッドプロシジャ名を定義している時だけ実行します。

```
if { ¥[info exists stval($stid,grid_proc)¥] } {
        eval ¥$stval($stid,grid_proc)
}
```

1.27 join

join	リストの要素をつなげて文字列を作成します
形式	join list [joinString]
説明	
引数 list は正しい Tcl のリストでなければなりません。このコマンドは list の全ての要素を joinString で区切って並べた文字列を返します。引数 joinString のデフォルトはスペースです。	
使用例	

```
# st_op.psv から抜粋。
# リスト形式の引数を文字列化して、ldeflist 変数に代入する。
set ldeflist [join $args]
```

1.28 lappend

lappend	リストに変数を要素として追加します
形式	lappend varName value [value value ...]

説明

このコマンドは varName で指定する変数をリストと見なし、そのリストに value で指定する各引数をそれぞれリストの要素として、スペースで区切って追加します。 varName が存在しなければ、value の各引数を要素とするリストとして新たに作成します。 lappend は append に似ていますが、各 value が単なるテキストではなく、リストの要素として追加するところが異なります。このコマンドを使うと大きなリストを効率的に作ることができます。たとえば ``set a [concat $a [list $b]]'' よりも``lappend a $b'' の方が $a が長い時には効率的です。

使用例

#gsamp107.psv から抜粋。乱数で XY 座標を生成し、座標のリストを生成するサンプル。

```
set plist ""
for {set lp 0} {$lp < $pcnt} {incr lp} {
          set x [expr "rand() * $gs107(wid)"]
          set y [expr "rand() * $gs107(hei)"]
          lappend plist $x $y
}
```

1.29 lindex

lindex	リストから要素を取り出します
形式	lindex list index
説明	
このコマンドは list を Tcl のリストと見なし、index 番目(0 が最初の要素) の要素を返します。要素を取り出す時に、lindex は中かっこやクォートやバックスラッシュに関して Tcl コマンドインタプリタと同じルールに従います。ただし、変数置き換えとコマンド置き換えは行なわれません。index が負の時や、index の要素数以上であった時には空文字列が返されます。	
使用例	
# print.psv サンプルから抜粋。プリンタ名のリストから先頭のプリンタ名を取り出し、代入する処理 set workList [split $pnamelist ","] set workVar [lindex $workList 0] set prt(p_name) $workVar	

1.30 linsert

linsert	リストに要素を挿入します
形式	linsert list index element [element element ...]

説明

このコマンドは list の index 番目の要素を指定して、その直前に全ての element 引数を挿入し、新しいリストを作成します。新しいリストの中で引数 element はそれぞれ独立した要素になります。index が 0 以下の時は、新しい要素をリストの先頭に挿入します。index がリストの要素数以上の時は、新しい要素をリストの最後に追加します。

使用例

```
# グラフィックスウィンドウのリストに、$gwin のグラフィックスウィンドウを挿入しています。
if { [llength $SAMP1(gwin_list)] > $idx } {
        set SAMP1(gwin_list) [linsert $SAMP1(gwin_list) $idx $gwin]
        ...
} else {
        lappend SAMP1(gwin_list) $gwin
}
```

1.31 list

list	リストを作成します
形式	list [arg arg ...]

説明

このコマンドは全ての arg を要素としてリスト作成して、返します。arg を一つも指定しない時には空の文字列を返します。結果のリストに対して lindex コマンドで元の引数を再び取出す時や、eval で最初の arg をコマンド名、残りをその引数として実行可能な様に、リストには必要に応じて中かっこやバックスラッシュを追加します。list が返すリストは concat のものとは多少異なります。concat はリストを作る前に引数がグループ化されている時は、そのグループを 1 レベル取りさるのに対し、 list はもとの引数をそのまま使用します。

たとえば

```
list a b {c d e} {f {g h}}
```

は

```
a b {c d e} {f {g h}}
```

を返しますが、同じ引数に対して concat を適用するとこうなります。

```
a b c d e f {g h}
```

使用例

```
# ２つのリストを１つのリストとして作成し、リスト要素でループしています。
set work [list $workbuf1 $workbuf2]
foreach i $work {
…
}
```

1.32 llength

llength	リストの要素を数えます
形式	llength list
説明	
list をリストと見なして、その要素の数を 10 進の文字列で返します。	
使用例	

```
# st_op.psv から抜粋、座標のリストが７個の時だけ特定のコマンドを実行する処理
if { [llength $gpoint] == 7  } {
          eval $stval($stid,ucom)
}
```

1.33 lrange

lrange	リストから連続する要素を取り出します
形式	lrange list first last
説明	
list はリスト構造の変数を指定します。このコマンドは first 番目から last 番目（first と last を含む）までの要素からなる新しいリストを作成して返します。 last には"end"も指定可能で、その場合それはリストの最後の要素を指します。 first が 0 より小さいなら、それは 0 として扱います。 last がリストの要素数以上ならそれは end として扱います。 first が last より大きければ空文字列を返します。	
使用例	
# gsamp090.psv から抜粋。図形オブジェクトの範囲座標のリストから、左下座標と右上座標を切りだす処理 set aaa [eval g_region_obj $gs090(objs)] eval g_draw_rect $gs090(gc) [lrange $aaa 0 1] [lrange $aaa 4 5] -c 3	

1.34 lreplace

lreplace	リストの要素を置き換えます
形式	lreplace list first last [element element ...]

説明

lreplace は list の要素を element 引数で置き換えたリストを作成し返します。first で list の置き換えるべき先頭の要素のインデックスを指定します。first が 0 より小さいなら、0 として扱います。first が指している要素はリスト中に必ず存在しなければなりません。 last で list の置き換えるべき最後の要素のインデックスを指定します。これは first 以上の値を指定してください。last として"end"も指定可能で、その場合 first からリストの最後までが置き換えの対象になります。引数 element で指定リストの要素と置き換えるための 0 個以上の新しい要素を指定します。各 element はそれぞれ独立した要素になります。引数 element の指定が無い場合、first と last の間の要素を削除したリストを作成します。

使用例

```
# st_sub.psv から抜粋。座標のリストから、ループで位置をずらしながら要素を置き換えている処理
for { set i 0 } { $i < $pnum } { incr i } {
        set cd [lreplace $cd [expr $i * 2] [expr $i * 2 + 1] ¥
        [expr [lindex $cd [expr $i * 2]] + $dx] ¥
        [expr [lindex $cd [expr $i * 2 + 1]] + $dy]]
}
```

1.35 lsearch

lsearch	リストに指定する要素があるかどうかを調べます
形式	lsearch [mode] list pattern

説明

このコマンドは list の中の pattern にマッチする要素を調べます。見つかれば最初にマッチした要素のインデックスを返します。見つからなければ -1 を返します。

引数 mode でリストの要素と pattern とのマッチング方法を指定できます。指定できるのは次の通りです。

-exact

リストの要素は pattern と全く同じ文字列でなければなりません。

-glob

pattern ワイルドカード文字を使ったパターンであり、リストの要素とは string match コマンドと同じ規則でマッチするかどうか調べます。

-regexp

pattern は正規表現であり、リストの要素とは regexp コマンドと同じ規則でマッチするかどうか調べます。

mode が省略した時は -glob の方式が使用されます。

使用例

#st_op.psv から抜粋。リスト内に$stid と同じ値の要素があるか調べ、無ければエラーとする処理

```
if { ¥[set num ¥[lsearch ¥$stval(stids) ¥$stid¥]¥] < 0 } {
        st_err ¥"error: <ST_ERROR> stid error¥"
        return ¥"error: <ST_ERROR> stid error¥"
}
```

1.36 lsort

lsort	リストの要素をソートします
形式	lsort [switches] list

説明

このコマンドは list の要素をソートし、新たなリストとして返します。デフォルトでは文字コード昇順のソートが行なわれます。しかし、次のような switches を list の前に指定することによってソート処理方法を変更可能です。

-ascii

文字コードの昇順でソートします。これがデフォルトです。

-integer

各要素を整数に直し、整数値の比較をします。

-real

各要素を実数に直し、実数値の比較をします。

-command command

command を比較用のコマンドに使います。 2 つの要素を比較するために command のスクリプトを 2 つの要素を引数として実行します。スクリプトは整数値を返すもので、最初の引数が 2 番目の引数より小さいならば 0 より小さい値を、等しいならば 0 を、大きいならば 0 より大きい値を返すようなものでなければなりません。

-increasing

リストを昇順（「最小」のものが最初）でソートします。これがデフォルトです。

-decreasing

リストを降順（「最大」のものが最初）でソートします。

使用例

```
#st_op.psv から抜粋。stval の要素名をリストとして取得し、名前昇順でソートしてからループで使用する
foreach i [lsort [array name stval]] {
          if { [regexp $stid $i] } {
              unset stval($i)
          }
}
```

1.37 open

open	ファイルをオープンします
形式	open fileName [access] [permissions]
説明	

このコマンドはファイルをオープンし、read、 puts、close コマンド指定する識別子を返します。 fileName でオープンするファイル名を指定します。fileName が ``|'' ではじまっていれば、 exec と同じように残りの文字はパイプラインで実行するコマンドと見なします。この場合、 open が返す識別子を使ってコマンドの入力パイプに書き込んだり、出力パイプから読み込んだりできます。

引数 access でファイル（またはコマンドパイプライン）へのアクセス形式を指定します。これには 2 種類の指定方法があります。一つはライブラリの fopen 関数に渡す形式の文字列で、もう一つは POSIX のアクセスフラグのリストです。デフォルトは ``r'' です。最初の指定方法では次のような指定が使えます。

r

ファイルを読み取り専用でオープンします。あらかじめファイルが存在していなければなりません。

r+

ファイルを読み取りと書き込みでオープンします。あらかじめファイルが存在していなければなりません。

w

ファイルを書き込み専用でオープンします。ファイルが既に存在していれば、既存ファイルの内容を破棄して先頭から書き込みます。なければ新たに作成します。

w+

ファイルを読み取りと書き込みでオープンします。ファイルが既に存在していれば、既存ファイルの内容を破棄して先頭から書き込みます。なければ新たに作成します。

a

ファイルを書き込み専用でオープンします。あらかじめファイルは存在していなくてはなりません。また書込んだデータはそのファイルの最後に追加します。

a+

ファイルを読み取りと書き込みでオープンします。ファイルが存在しなかった時には、新たに空のファイルを作成します。またファイルのアクセスはそのファイルの最後から行なわれます。

2 番目の指定方式では access には次にあげるフラグのリストを指定します。これらはすべて POSIX で定義するマクロ値と同じ意味を持っています。少なくとも RDONLY、 WRONLY、 RDWR のどれか一つを指定しなくてはなりません。

説明

RDONLY

ファイルを読み取り専用でオープンします。

WRONLY

ファイルを書き込み専用でオープンします。

RDWR

ファイルを読み取りと書き込みでオープンします。

APPEND

書き込みのたびに、それに先立ってファイルポインタをファイルの最後に設定しなおすようにします。

CREAT

ファイルが存在しない場合には新たに作成します(このフラグがないと、ファイルが存在しない場合はエラーになります)。

EXCL CREAT

フラグとともに指定すると、ファイルが存在している場合にエラーになります。

NOCTTY

ファイルが端末デバイスだった場合、このフラグを指定するとそのファイルがプロセスの制御端末にならないようにします。

NONBLOCK

ファイルのオープンの際にプロセスがブロックしないようにします。詳しくは open システムコールのマニュアルの O_NONBLOCK フラグに関する説明を見てください。

TRUNC

ファイルが存在する場合、サイズを 0 に切り捨てます。

オープン時に新たなファイルを作成する場合、 permissions がプロセスのファイルモード生成マスクとともに、新たに作成するファイルのパーミッションを設定します。 permissions デフォルトは 0666 です。

※PreServ の Tcl では permissions の指定は意味がないため、指定を省略してください。

ファイルが読み取りと書き込みの両方でオープンしている時、読み取りと書き込みを切り替える時には seek を実行します(この制限は open でコマンドパイプラインをオープンした時には適用されません)。

fileName にコマンドパイプラインを指定し、書き込み専用でオープンした場合には、パイプラインの標準出力は、そのコマンドによってリダイレクトしていないかぎりアプリケーションの標準出力に出力します。 fileName にコマンドパイプラインを指定し、読み取り専用でオープンした場合には、パイプラインの標準入力には、そのコマンドによってリダイレクトしていないかぎりアプリケーションの標準入力を使用します。

使用例

color.psv から抜粋。ファイルを書き込みモードでオープンし、行取得の gets コマンドを実行する処理

```
    if { [catch { set fd [open $filename r] } str] != 0 } { return -1 }
    if { [catch { gets $fd macro } str] != 0 } { return -1 }
```

1.38 pid

pid	プロセス ID を取得します
形式	pid [fileId]
説明	
引数 fileId を指定する時は、通常 open コマンドのパイプラインのファイル識別子を指定します。この場合 pid コマンドはパイプラインの全プロセスのプロセス ID をリストとして返します。もし fileId がパイプライン以外のものを指している場合には空のリストを返します。 fileId が指定しない時は、pid はカレントプロセス(WebTcl エンジンプロセス)のプロセス ID を返します。プロセス ID はすべて 10 進の文字列で返します。	
使用例	
－	

1.39 proc

proc	Tcl プロシジャを作成します
形式	proc name args body

説明

proc コマンドは name という名前のプロシジャを作ります。同じ名前のコマンドやプロシジャが既に存在する場合は、このプロシジャで上書きします。この name プロシジャを起動すると、body の内容を実行します。

args ではこのプロシジャに対する引数を指定します。args はプロシジャの引数をリスト形式で指定します（使用例参照）。

各引数指定もリストで指定可能です。各引数は 1 つまたは 2 つの要素を持ちます。要素が 1 つであればそれは引数の名前になります。2 つなら最初が引数名で 2 番目がそのデフォルト値です（使用例参照）。

name プロシジャを起動すると、プロシジャの各引数指定で定義した引数名の名前を持つローカル変数を作ります。変数の値はコマンドに渡す引数の値か、デフォルトの値となります。デフォルト値を指定する引数については、コマンド起動時に省略可能です。しかし、デフォルトを指定しない引数に対しては起動時省略できません。また引数指定で定義する引数より多くの引数を指定するとエラーになります。

引数の数が不定のプロシジャを実現するためには特別な書式が必要です。引数指定で定義する最後の引数の名前が args であると、そのプロシジャに対して引数指定に定義するよりも多くの引数を渡すことができます。この場合、本来 args に渡すべき引数からあとの引数を 1 つのリストにまとめます（list コマンドのように）。このまとめた値をローカル変数 args にセットします。

body を実行する時には変数名は通常ローカル変数を指します。ローカル変数は参照時に自動的に作り、プロシジャから抜ける時に自動的に消します。プロシジャの引数ごとにローカル変数を自動的に作ります。グローバル変数を参照するには global コマンドか upvar コマンドを使わなくてはなりません。

proc コマンドは空文字列を返します。name プロシジャを起動した時、そのリターン値は return コマンドで指定した値になります。もしプロシジャが明示的に return を呼ばなかった場合、そのリターン値はプロシジャの本体で最後に実行したコマンドの値になります。もしプロシジャ本体の実行中にエラーが起こったならば、プロシジャ自身もそのエラーでリターンします。

使用例

```
# st_op.psv から抜粋。引数の変数名をリスト形式で指定している例
proc st_operation { stid gc gw dl ope args } {...}
# 引数自体がリストで、デフォルト値を定義し、引数 b をグローバル変数 chk に代入する処理
proc testProc01 { a {b 1} } { global chk; set chk $b}
# ２つとも引数を定義した場合
testProc01 x y
→$chk は"y"が入っている
# １つだけ引数を定義した場合
testProc01 x
→$chk は"1"が入っている
# 可変個数の引数の定義例
proc testProc02 { a args} { global chk; set chk $args}
# 引数は１～ｎ定義可能
testProc02 x
testProc02 x y z a b c
→この場合$chk には{ y z a b c }が入っている
# 比較として引数１or２のプロシジャを実行
testProc01 x y z a b c
→「called "testProc01" with too many arguments」のエラーとなる
```

1.40 puts

puts	ファイルに一行書き込みます
形式	puts [-nonewline] [fileId] string

説明

fileId で指定するファイルに string で指定する文字列を書き込みます。fileId は open が返す値か、標準入出力に割り当てている stdout または stderr を指定してください。また、ファイルは書き込み用にオープンしてください。fileId を省略した時にはデフォルト値の stdout を使用します。通常 puts は string のあとに改行コードを出力しますが、この機能は引数-nonewline を指定することによって解除できます。ファイルへの出力は Tcl の内部でバッファリングします。flush コマンドでバッファリングした文字列を強制的に出力させることができます。

使用例

```
# color.psv から抜粋。色を定義するマクロを puts で書き出している処理
  proc col_write { filename } {
    variable pb
    if { [catch { set fd [open $filename w] } str] != 0 } { return -1 }
    set str ""
    set work "proc psv_set_color {gc} {"
    catch { puts $fd $work  } str
    if { $str != "" } { return -1 }
    for {set i 1} {$i <= 255} {incr i} {
      set bg [w_get_value $pb($i) -b]
      if { [catch { puts $fd  "¥tg_alloc_color ¥$gc $i $bg" } str] != 0 } {
        return -1
      }
    }
    set str ""
    set work "}"
    catch { puts $fd $work  } str
    if { $str != "" } { return -1 }
    if { [catch { close $fd } str ] != 0 } { return -1 }
    return 0
  }
```

1.41 pwd

pwd	現在の作業ディレクトリを返します
形式	pwd
説明	
現在の作業ディレクトリ（カレントワーキングディレクトリ）のパス名を返します。 WebTcl では、物理的なディクトリ移動ができないため、AP サーバ上のコンテンツのパス名を返します。 本コマンドは A 版時に作成したスクリプトの互換性のために仕様を調整しています。 相対パス概念のエミュレートでスクリプトの互換性を保つ用途に制限して使用してください。	
使用例	
－	

1.42 read

read	ファイルからの読込みを行います
形式	read [-nonewline] fileId read fileId numBytes
説明	
最初の形式では、fileId で指定するファイルの残りのデータ全てを読込み、このコマンドの結果として返します。もし-nonewline を指定していれば、ファイルの最後の文字が改行コードなら改行コードを読み捨てます。 2 番目の形式では、何バイト読むかを指定する形式です。ファイルの残りのデータが numBytes バイトより少なければ残りのデータ全てを読出します。fileId は stdin または open が返した値でなければなりません。また、ファイルは読み取り用にオープンしてください。read コマンドは実行に先立ってファイルの EOF やエラー状態のフラグをクリアします。	
使用例	
―	

1.43 regexp

regexp	文字列と正規表現のマッチングを行います
形式	regexp [switches] exp string [matchVar] [subMatchVar sub-MatchVar ...]

説明

正規表現 exp が string の全体または一部にマッチするか調べ、マッチすれば 1、マッチしなければ 0 を返します。

string の後の引数が指定した場合、それらは string のどの部分が exp にマッチしたかという情報を返すための変数名として扱います。matchVar には string の exp 全体がマッチした範囲をセットします。subMatchVar には exp の中の最も左側にあるかっこで括った、副表現にマッチする文字列をセットし、次の subMatchVar にはその次のかっこで括った副表現にマッチする文字列をセットします。

regexp の最初の引数が-で始まる時は、オプションスイッチと見なします。オプションは以下のものを指定可能です。

-nocase

マッチングの時、string 中の大文字を小文字とみなして処理します。

-indices

subMatchVar に入れられる内容を変えます。string のマッチした部分の文字の代わりにそれぞれの変数には string の中でマッチした部分の最初と最後の文字のインデックスをそれぞれ 10 進文字列で表したリストをセットします。

--

オプションの終りを示します。これ以降の引数はたとえそれが-で始まっていたとしても exp と見なします。

exp の中のかっこで括った副表現の数より subMatchVar の数の方が多い時や、exp の副表現の中にマッチしなかったものがある時には、マッチしなかった副表現に対応する subMatchVar には-indices を指定した時は``-1 -1''が、指定が無い場合は空文字列をセットします。

正規表現

正規表現とは 0 個またはそれ以上の``|''で分けた「枝(branches)」からなります。枝のどれかとマッチすればその正規表現とマッチしたことになります。

枝とは 0 個またはそれ以上の「ピース (pieces)」がつながったものです。最初の部分が最初のピースとマッチし、その次の部分が 2 番目のピースとマッチし…となっていればその枝とマッチしたことになります。

ピースとは「アトム (atom)」またはアトムのあとに ``*'' か``+'' か ``?'' がついたものです。アトムのあとに ``*'' がついたものはそのアトムに 0 個以上連続してマッチするものにマッチします。アトムのあとに ``+'' がついたものはそのアトムに 1 個以上連続してマッチするものにマッチします。アトムのあとに ``?''がついたものはそのアトムにマッチするものか、または空文字列にマッチします。

アトムとは、かっこで括った正規表現(その正規表現にマッチするものとマッチ)、「範囲 (range)」 (下を参照のこと)、 ``.''(任意の文字とマッチ)、 ```^'' (入力文字列の先頭の空文字列にマッチ)、 ``$'' (入力文字列の最後の空文字列にマッチ)、 ``¥''とそれに続く文字 (その文字とマッチ)、その他の文字 (その文字とマッチ) のいずれかです。

説明

「範囲 (range)」とは文字の並びを ``[]'' で括ったものです。普通はその並びの中にある任意の文字とマッチします。もしも並びが``^'' ではじまっていれば、その並びの中にない任意の文字とマッチします。もしも並びの中に``-''でつなげた 2 文字があれば、それはその 2 文字の間にある全 ASCII 文字のリストの省略記法です (たとえば``[0-9]''は任意の 10 進数字にマッチします)。文字の並びの中に``]''を含めたい時にはそれを並びの最初の文字 (``^'' があればそのあと)にしてください。``-'' を含めたい時には最初か最後の文字にしてください。

マッチング方法の選択

一般的にいって、ある入力文字列に対する正規表現のマッチングの方法が何種類もあることがあります。例えば次のコマンドをみてください。

```
regexp (a*)b* aabaaabb x y
```

今まで述べた規則だけで考えると、x と y は aabb と aa、aaab と aaa、ab と a などいくつかの組合せと解釈できます。この解釈のあいまいさをなくすため、regexp はいくつかの選択肢の中から「最初の、そして最長の」という規則で選び出します。別の言い方をすると、マッチングを調べるに当たって入力文字列の左から右に向かって順に調べていき、さらに入力文字列のマッチする部分が長いものを短いものより先に調べていきます。具体的には以下のような規則に従います。規則は優先度の高いものから順に並べてあります。

[規則 1] 正規表現が入力文字列の 2 つの異なる部分にマッチ可能な場合、そのマッチする部分の始まりが文字列の先頭により近い方にマッチします。

[規則 2] 正規表現に | オペレータを含む場合にはもっとも左側の副表現を選択します。

[規則 3] *、+、?を使った表現ではマッチする部分が長いものの方を短いものより優先します。

[規則 4] いくつかの部分表現が並んでいるような場合には、左側の部分表現からマッチングを調べます。

前述の例では (a*)b* は aab にマッチします。パターンの(a*) という部分が最初にマッチし、これが最初の aa という部分とマッチします。次に b* の部分が次にくる b にマッチします。

次の例も確認してみます。

```
regexp (ab|a)(b*)c abc x y z
```

このコマンドを実行すると x は abc に、y は ab に、そして z は空文字列になります。規則 4 により(ab|a)を最初に調べ、規則 2 によって a という副表現より先に ab という副表現が先に調べます。したがって b は正規表現の(b*)という部分が調べられる前にすでにその前の部分にマッチしてしまっているので(b*)はマッチする対象がないということになります。

使用例

```
# filese.psv から抜粋。拡張子を大文字小文字区別無しでマッチングを行う処理
          if { [file isdirectory $fn] } {
                    lappend lld "[file tail $fn]"
#                   拡張子判別処理で大文字小文字区別しない
          } elseif { $FSval($id@ex) == "" ||
           [regexp -nocase "¥.$FSval($id@ex)" [file extension $fn] ] == 1 } {
                    lappend llf "[file tail $fn]"
          }
```

1.44 regsub

regsub	正規表現のパターンマッチングに基づいて置換えを行います
形式	regsub [switches] exp string subSpec varName

説明

このコマンドは正規表現 exp を string とマッチさせ、string を varName で指定される変数にコピーします。マッチすれば 1 を、しなければ 0 を返します。マッチして string を varName にコピーする時に string 中の exp にマッチした部分を subSpec で置き換えます。

置き換えには以下の規則を適用します。

[規則 1] subSpec に``&''または``¥0'' を含む時は、それは string の exp とマッチした部分で置き換えます。

[規則 2] subSpec に``¥n''(ただし n は 1 から 9 までの数字)を含む時は、それは string 中で、exp の n 番目のかっこで括った副表現とマッチした部分で置き換えます。

subSpec 中の``&''や``¥0''や``¥n''やバックスラッシュに特別な意味を持たせないようにするにはそれらの前にバックスラッシュをつけます。ただし subSpec の中に書いたバックスラッシュは Tcl インタプリタでも解釈の対象となるので、subSpec の中にバックスラッシュを入れる時には中かっこで括ることをお奨めします。

regexp の最初の引数が-で始まっている場合、それはオプションスイッチと見なします。次のようなオプションを使用することができます。

-all

exp にマッチした string のすべての部分に対して置き換えが行なわれます。このオプションを指定しない時には最初にマッチした部分に対してしか置き換えは行なわれません。-all を指定した時には``&''と``¥n''はマッチした部分ごとに扱います。

-nocase

exp とマッチングを行う前に string の中の大文字を小文字に変換します。しかし、subSpec で指定する置き換えは string の変換前の文字列をそのまま使って行います。

--

オプションの終りを示します。これ以降の引数はたとえそれが-で始まっていたとしても exp であると見なします。

正規表現の解釈についての詳細は regexp コマンドを参照してください。

使用例

#color.psv から抜粋。”&XXXXXX”の表記を”0xXXXXXX”の一般的な 16 進表記に置き換える処理

```
proc col_rgb { rgb red green blue } {
  upvar $red r $blue b $green g
  regsub "&" $rgb "0x" rgb
  set r [expr ($rgb >> 16) & 0xff]
  set g [expr ($rgb >> 8 ) & 0xff]
  set b [expr ($rgb      ) & 0xff]
}
```

1.45 rename

rename	コマンドの名前の変更・削除を行います
形式	rename oldName newName
説明	
oldName という名前のコマンドを newName という名前に変えます。もし newName が空文字列(たとえば {})であった場合には oldName のコマンドは削除します。rename は結果として空文字列を返します。	
使用例	
# gsamp014.psv から抜粋。終了時にプロシジャを削除する処理 rename gs014p ""	

1.46 return

return	プロシジャからリターンします
形式	return [-code code] [-errorinfo info] [-errorcode code]

説明

valueをリターン値として現在のプロシジャ(もしくはトップレベルコマンドかsourceコマンド)から直ちにリターンします。valueを指定しない時は、空文字列を返します。

例外時の戻り値

-codeを指定しない時は、プロシジャを正常終了します（終了コードはTCL_OK）。

-code指定する時は、例外を発生させてプロシジャから異常終了します。codeには次のような値を使用することができます。

ok

正常なリターン。オプション省略時と同じです。

error

エラーリターン。errorコマンドでプロシジャを異常終了ほぼ同じ動作ですが、後述のerrorInfoとerrorCode変数の扱いが異なります。

return

現在のプロシジャから終了コードTCL_RETURNでリターンします。これによりこのプロシジャを起動したプロシジャからもリターンすることになります。

break

現在のプロシジャから終了コードTCL_BREAKでリターンします。これによってこのプロシジャを起動したコードの最も内側のループの実行が終ることになります。

continue

現在のプロシジャから終了コードTCL_CONTINUEでリターンします。これによってこのプロシジャを起動したコードの最も内側のループで、ループの残りの実行をスキップします（つまり次のループ実行に移ることになります）。

value

valueは整数でなければなりません。これをプロシジャの終了コードとして使用します。

-codeオプションはアプリケーションで特殊な制御構造を実現する場合、呼び出し元に例外条件を渡すための機能です。一般的なアプリケーションで使用する必要はありません。

-errorinfoと-errorcodeというオプションは、エラーでリターンする時にエラー情報を指定することができます。この2つのオプションはcodeがerrorの時以外は無効です。

説明

-errorinfo オプションは errorInfo 変数に格納するスタックトレースの初期値を指定するためのものです。指定しない時は、errorInfo に入るスタックトレース情報には現在のプロシジャと呼び出し側プロシジャの情報が入ります。ただし現在のプロシジャ内でのエラーのコンテキストについての情報は入りません。

一般的な用途では info にはそのプロシジャ内で catch コマンドによってトラップしたエラーの errorInfo を指定します。

-errorcode オプションを指定した時は、code を errorCode 変数の値として使います。指定しない時は errorCode の値はデフォルトの NONE になります。

使用例

#color.psv から抜粋。RGB 値をフォーマットして返しています。

```
proc col_bg { red green blue } {
        return [format "&%2.2x%2.2x%2.2x" $red $green $blue]
}
```

1.47 scan

scan	sscanf のスタイルの変換指定子を使用して文字列を変換します
形式	scan string format varName [varName ...]

説明

このコマンドは C 言語ランタイムの sscanf と同様に入力文字列を解釈し、正しく解釈できたフィールドの数を返します。string で入力文字列を指定し、format で解釈時のデータ型を sscanf と同様に% 変換指定子を用いて指定します。varname にはそれぞれ変数名を指定します。string から取り出したフィールドは文字列の形式に再変換し、対応する varname にセットします。

scan の仕様詳細

scan は string と formatString を対応させながらスキャンすることにより処理を行います。formatString 中のスキャンする次の文字がスペースまたはタブならばそれは無視します。formatString 中に%が出てくると、それは変換指定子の開始と見なします。変換指定子は%とそれに続く 3 つのフィールドから構成されます。最初に*があると、それは変換した値を変数に代入するのではなく、捨てるべきであることを表します。次にフィールドの最大幅を表す数字がきて、最後に変換指定文字がきます。これらの 3 つのフィールドのうち、変換指定文字以外は省略可能です。

formatString 中に変換指定子を見つけると、scan はまず string の空白文字をスキップします。次に変換指定にしたがって入力データを変換し、その結果を scan の引数で指定する変数に次々に代入していきます。次のような変換指定文字を用意しています。

d

入力フィールドは 10 進の整数でなくてはなりません。読み込んだデータは 10 進の文字列として変数に代入します。

o

入力フィールドは 8 進の整数でなければなりません。読み込んだデータは 10 進の文字列として変数に代入します。

x

入力フィールドは 16 進の整数でなければなりません。読み込んだデータは 10 進の文字列として変数に代入します。

c

1 文字読み込んでその文字コードを 10 進の文字列として変数に代入します。この場合には空白文字のスキップは行なわれません。したがって入力フィールドが空白文字のこともあります。この変換は、入力フィールドが常に 1 文字であり、フィールド幅を指定してはいけないという点が C 言語ランタイムの scanf と異なります。

s

入力フィールドは次の空白文字までのすべての文字です。その文字列を変数にコピーします。

説明

e or f or g

入力フィールドは実数を指定してください。実数は符号(省略可能)、10 進数字の列(小数点があってもよい)、指数部(これも省略可能)から構成されます。指数部は e または E とそれに続く符号 (省略可)と 10 進の数字からなります。読み込んだデータは実数の文字列として変数に代入します。

[chars]

入力フィールドは chars に含む任意の文字の並びです。マッチした文字列が変数に入ります。開き大かっこの次の文字が]である時は、chars の一部だとみなし、指定の最後の閉じかっこにはなりません。

[^chars]

入力フィールドは chars を含まない任意の文字の並びです。マッチした文字列が変数に入ります。^の次の文字が]である時は、それは chars の一部だとみなし、指定の最後の閉じかっこにはなりません。

ある変換に対して読み込む文字は、その変換が意味をなすもののなかで最大文字数のものになります (例えば%d の場合 10 進数が続くかぎりになりますし、%o の場合 8 進数が続くかぎりになります) 。指定した変換に対する入力フィールドは、空白文字が出てくるかフィールドの最大幅に達するかまでの範囲です。変換指定子に*が入っていれば変数への代入は起きず、scan の引数も使いません。

C 言語ランタイムの scanf との違い

scan コマンドの動作は C 言語ランタイムの sscanf 関数と次の点の動作が異なります。

[1] %p と %n 変換はサポートしません。

[2] %c 変換では 1 文字の文字コードが 10 進文字列に変換し、対応する varName に代入します。フィールド幅を指定することはできません。

[3] l、h、L 修飾子は無視します。整数値は常に修飾子なしの状態で変換が行なわれ。実数値は常に l 修飾子がついた状態(つまり内部表現として double が使われます)で変換されます。

使用例

```
# color.psv から抜粋。ファイルから１行入力し、スペースで区切られた情報から４つの文字列を取り出す処理
if { [catch { gets $fd macro } str] != 0 } {
        close $fd
        return -1
}
scan $macro "%s %s %s %s" dm1 dm2 no rgb
# gsamp003.psv から抜粋。マウスイベント座標から WC 座標を求め、２つの浮動小数点値として取り出す例
w_get_event
scan [g_dc2wc $gs003(gc) $psv_event(x) $psv_event(y)] "%f %f" gs003(x) gs003(y)
```

1.48 seek

seek	オープンしたファイルのアクセス位置を変更します
形式	seek fileId offset [origin]

説明

fileId のアクセス位置を変更します。

注）WebTcl でネットワーク上のファイルをダウンロードする場合は、この機能は無効です。

fileId は open コマンドが返した値か、あるいは標準入出力チャネルを示す stdin、stdout、stderr のいずれかを指定してください。引数 offset と origin で fileId に対する次の読み書きの開始位置を指定します。offset は整数(負でもよい)で指定し、origin は以下の値を指定します。

start

アクセス位置はファイルの先頭から offset バイト目に設定します。

current

アクセス位置は現在のアクセス位置から offset バイトの所に移動します。offset が負なら、アクセス位置はファイルの前方に移動します。

end

アクセス位置はファイルの最後から offset バイトの場所になります。offset が負ならアクセス位置はファイルの最後よりは手前になり、正ならファイルの最後より後になります。

引数 origin のデフォルトは start です。このコマンドは空文字列を返します。

使用例

－

1.49 set

set	変数を設定・取得します
形式	set varName [value]

説明

value 省略時は、変数 varName の値を返します。value を指定する時は、varName の値を value にセットして、その値が返ります。もし変数 varName が存在していなければ新たに作ります。もし varName に開き小かっこが含み、閉じ小かっこで終っていればそれは配列の指定になります。開きかっこの前の文字列が配列名になり、かっこ内の文字列が配列のインデックスになります。それ以外の場合には varName は通常の変数で指定します。もしプロシジャ外で実行する場合、varName はグローバル変数の操作となります。プロシジャ内で実行する場合、varName はプロシジャの引数かローカル変数になります。global コマンドで varName がグローバルと宣言している時はプロシジャ内でもグローバル変数の操作となります。

使用例

```
# color.psv から抜粋。通常の変数の設定を実行している例
proc col_rgb { rgb red green blue } {
        upvar $red r $blue b $green g
        regsub "&" $rgb "0x" rgb
        set r [expr ($rgb >> 16) & 0xff]
        set g [expr ($rgb >> 8 ) & 0xff]
        set b [expr ($rgb      ) & 0xff]
}
# color.psv から抜粋。配列 col のインデックス top_t の要素に値を設定している例
set col(top_t)      "色定義ダイアログ"
```

1.50 source

source	ファイルを Tcl スクリプトとして実行します
形式	source filename

説明

ファイル filename を読み、その中身をコマンド列として Tcl インタプリタに送り実行します。source のリターン値はファイルで最後に実行したコマンドのリターン値になります。ファイルの内容の実行中にエラーが起きれば、source コマンドは残りのコマンドの実行をスキップし、エラーを返します。ファイルの中で return コマンドを実行するとファイルの残りの実行をスキップし、source の実行は正常に終了します。リターン値は return コマンドの仕様に従います。

WebTcl では、FE-COM.codeBase プロパティの値を filename 先頭に付加して、Web サーバの URL としてファイルをダウンロードし、実行します。

使用例

```
# gindex.psv から抜粋。ボタンを押すと対応する G マクロサンプルを source する処理
set Gl(top) [w_top Gl(top1) -x 800 -y 0 -w 224 -h 800]
w_button Gl(btn001) $Gl(top) -x   0 -y   0 -w 110 -h 20 -v g6 -l gsamp001.psv -c {source gsamp001.psv}
```

1.51 split

split	文字列を分割して Tcl のリストを作成します
形式	split string [splitChars]

説明

string を引数 splitChars に含まれる文字の位置で分割し、リストを作成して返します。string の、splitChars と一致する文字列ではさまれた部分の文字列が結果のリストの各要素になります。string の中に splitChars と一致する文字列が連続した場合、string の最初または最後の文字が splitChars と一致する文字列であった場合には、対応する要素部分が空のリスト要素となります。splitChars が空文字列の場合、結果は string の各文字がリスト要素となります。splitChars の指定省略時はデフォルト値の空白文字になります。

たとえば

split "comp.unix.misc" .

は "comp unix misc" を返し、

split "Hello world" {}

は "H e l l o { } w o r l d" を返します。

使用例

```
# print.psv から抜粋。CSV 形式の$pnamelist を”,” で分割し、先頭の要素を取り出している処理
set workList [split $pnamelist ","]
set workVar [lindex $workList 0]
```

1.52 string

string	文字列を操作します
形式	string option arg [arg ...]

説明

option により各種文字列操作を行います。以下に option に指定できる操作の説明を行います。

string compare string1 string2

C 言語ランタイムの strcmp と同様に、文字列 string1 と string2 に対して文字ごとの比較を行います。string1 が辞書順で string2 より小さいか等しいか大きいかによってそれぞれ -1、0、1 を返します。

string first string1 string2

string2 の中に string1 が出現するかを調べます。見つかれば string2 の中の最初に出現した場所(先頭文字のインデックス)を返します。見つからない場合は-1 を返します。

string index string charIndex

string の charIndex 番目の文字を返します。charIndex の 0 は文字列の最初の文字に対応します。charIndex が 0 より小さい場合や、文字列の長さ以上だった場合には空文字列を返します。

string last string1 string2

string2 の中に string1 が出現するかを調べます。見つかれば string2 の中で最後に出現した場所(先頭文字のインデックス)を返します。見つからない場合は-1 を返します。

string length string

string の文字数を 10 進の文字列で返します。

string match pattern string

pattern が string にマッチするか調べ、マッチすれば 1 を、しなければ 0 を返します。次に述べる pattern の特殊なシーケンスを除けば 2 つの中身が全く同じである場合、二つの文字列がマッチしたと判定します。

*

string の任意の文字(空文字列を含む)にマッチします。

?

string の任意の 1 文字とマッチします。

[chars]

chars で指定する集合中の任意の文字とマッチします。chars に x-y という形式の並びがあった場合、x から y までの任意の文字(x、y も含む)という意味になります。

¥x

x とマッチします。これは pattern の中の*?[]¥をパターンとして指定する場合に使用します。

説明

string range string first last

string の、インデックスが first から last までの部分文字列を返します。インデックス 0 は文字列の最初の文字を示します。文字列の最後の文字を示すために last として end を指定できます。first が 0 より小さい時は 0 を指定したとみなします。また last が文字列の長さ以上の時には end を指定したとみなします。もし first が last より大きい時には空文字列を返します。

string tolower string

string の全ての大文字を小文字に直して返します。

string toupper string

string の全ての小文字を大文字に直して返します。

string trim string ?chars?

string の最初か最後に chars で指定する文字の並びがある場合は、その並びを削除した文字列を返します。なければ string をそのまま返します。chars を指定しない場合は、空白文字(スペース、タブ、改行とキャリッジ・リターン)を削除します。

string trimleft string ?chars?

string の最初に chars で指定する文字の並びがある場合は、その並びを削除した文字列を返します。なければ string をそのまま返します。chars を指定しない場合は、空白文字(スペース、タブ、改行とキャリッジ・リターン)を削除します。

string trimright string ?chars?

string の最後に chars で指定する文字の並びがある場合は、その並びを削除した文字列を返します。なければ string をそのまま返します。Chars を指定しない場合は、空白文字(スペース、タブ、改行とキャリッジ・リターン)を削除します。

使用例

```
# wsamp049.psv から抜粋。テキストボックスの文字列長を調べ。選択範囲に使用する処理
set ws049(to) [string length [w_text_str $ws049(tx1)]]
w_text_select $ws049(tx1) 0 $ws049(to)
# 文字列に”vwin,”という文字列がないか切り出して調べる処理。
if {[string range $i 0 4] == "vwin,"} {
          set res [psvwb_ViewExit [string range $i 5 end]]
          if {$res=="C"} {break}
} else {
          if {[string range $i 0 7] == "PDebwin,"} {
                    w_destroy $psvwb($i)
          }
}
```

1.53 switch

switch	条件分岐コマンド
形式	switch [options] string pattern body [pattern body ...] switch [options] string {pattern body [pattern body ...]}

説明

switch コマンドは pattern 引数について順に string とマッチを試みます。string とマッチする pattern が見つかると、pattern に対応する body を実行し、その結果を返します。最後の pattern が default であれば、無条件に deault に対応する body を実行します。どの pattern も string にマッチせず、また default も指定しない場合 switch は空文字列を返します。

switch の最初の引数が - で始まっていればそれはオプションだとみなされます。オプションは以下のものが使用できます。

-exact

string とパターンが完全に一致した場合だけマッチすると判断します。これがデフォルトです。

-glob

string match コマンドと同じ規則で pattern がマッチするか調べます。

-regexp

正規表現によるマッチングを行います。

--

オプションの終りを示します。この次の引数は、たとえそれが - で始まっていたとしても string とします。

pattern と body に関して 2 種類のシンタックスを用意しています。

一つは各パターンとコマンドに対して別々の引数を使うもので、この形式はパターンやコマンドに対して置き換えが必要な時に便利です。

もう一つは全てのパターンとコマンドを一つの引数にまとめるというものです。引数は要素がパターンとコマンドであるような適切なリスト構造である必要があります。この形式だと複数行に渡る switch コマンドを作るのが簡単になります。これはリストを囲む中かっこのおかげで各行の終りにバックスラッシュを入れる必要がないからです。 2 番目の形式では pattern 引数を中かっこで括るためコマンド置き換えや変数置き換えが行なわれません。このため最初の形式のものと動作が異なることがあります。もし body が ``-'' だった場合には、次のパターンに対応する body をこのパターンに対しても使う、という意味になります。もし次のパターンも ``-'' ならさらにその次…というふうになります。この機能を使えば複数のパターンで一つの body を共有することができます。

以下に switch コマンドの例をいくつか示します。

switch abc a - b {format 1} abc {format 2} default {format 3}

は 2 を返し、

説明

switch -regexp aaab {

^a.*b$ -

b {format 1}

a* {format 2}

default {format 3}

}

は 1 を返します。

また

switch xyz {

a

-

b

{format 1}

a*

{format 2}

default

{format 3}

}

は 3 を返します。

使用例

ー

1.54 tcl

tcl	Tcl 言語のシンタックスの概要
形式	
説明	

Tcl 言語の基本的な文法を以下で説明します。

[1] スクリプトとは一つ以上のコマンドを含む文字列のことです。セミコロンと改行がコマンドの区切りになります。ただし"¥"で改行をクォートしている場合は除きます。コマンド置換の時にはクォートしていない「大括弧閉じ」によってコマンドが終了します。

[2] コマンドの評価実行は 2 段階で行われます。

(a) Tcl インタプリタは最初にコマンドをワードに分割します。

(b) 分割後、以下に示すような置き換えを行います。

置き換えは全てのコマンドに対して同じように行われます。最初のワードによってそのコマンドを実行するためのプロシジャを決定して、そのプロシジャにコマンドのワード全部を渡します。プロシジャは渡したワードをそれぞれ独自のやりかたで解釈します。例えば整数、変数名、リスト、スクリプトなどとして解釈します。プロシジャによってワードの解釈の仕方が異なります。

[3] コマンドの中のワードは空白文字(スペースやタブなど)によって区切ります（改行文字は除きます。これはコマンドの区切りになります）。

[4] ワードの最初の文字がダブルクォートだった場合には、次のダブルクォート文字までがワードになります。クォートの中にセミコロンや閉じかっこ、それに空白文字(改行も含みます)が出てきても、それは普通の文字として扱い、ワードに含みます。クォート中の文字に対してはコマンド置換、変数置換、バックスラッシュ置き換えが行われます。ダブルクォート文字自身は置換時に除去し、置き換え後のワードには含みません。

[5] ワードの最初の文字が「中括弧開き」だった場合には、対応する「中括弧閉じ」までがワードになります。中括弧はワードの中でネストすることができます。ワード中にさらに「中括弧開き」が出てきたら、それに対応する「中括弧閉じ」が必須です（ただし、中括弧がバックスラッシュでクォートしている場合には、区切り符号と見なさず、ネストの処理対象外です）。

中括弧内の文字に対しては。以下に述べるバックスラッシュ-改行置き換えを除き、一切置き換えは行われませんし、セミコロン、改行、「大括弧閉じ」、それに空白文字に対して特別な解釈は行われません。置き換え後のワードには一番外側の中括弧で囲んだ文字がそのまま入ります。外側の中括弧はワードから除去します。

[6] ワードに「大括弧開き」を含む場合、Tcl インタプリタは、コマンド置き換えを行います。この置き換えをするために Tcl インタプリタで、「大括弧開き」に続く文字列をスクリプトとして実行します。「大括弧閉じ」までがスクリプトだとみなし、スクリプトには複数のコマンドを書くことができます。大括弧とそれに囲まれた部分をスクリプトの実行結果(複数ある時には最後のコマンドの実行結果)で置き換えます。一つのワードに複数のコマンド置き換えがあっても構いません。コマンド置き換えは中括弧囲まれたワードに対しては行いません。

説明

[7] ワードにドル記号(``$'')を含む場合には Tcl インタプリタは、変数置き換えを行います。ドル記号とそれに続く文字を変数の内容で置き換えます。変数置き換えには以下に示す形式があります。

$name

name が変数の名前です。name にはアルファベット、数字、アンダースコア、および日本語コード以外の文字を含めることはできません。

$name(index)

name が配列変数の名前です。index がその配列の要素名を表します。name にはアルファベット、数字、アンダースコア、および日本語コード以外の文字を含めることはできません。index の文字に対してコマンド置換、変数置換、バックスラッシュ置き換えが行われます。

${name}

name は変数の名前です。名前には閉じ中かっこ以外の任意の文字を含めることができます。

一つのワードに複数の変数置き換えがあっても構いません。変数置き換えは中かっこで囲まれたワードに対しては行いません。

[8] ワードにバックスラッシュ(``¥'')を含む場合にはバックスラッシュ置き換えが行います。以下に述べる場合を除いて、バックスラッシュを除去し、その次の文字が普通の文字としてワードに入ります。これによりダブルクォート、閉じ中かっこ、ドル記号などを特別な意味がある文字をワード中に含めることができます。次の表は特別に解釈するバックスラッシュのシーケンスと、その置き換え内容です。

¥a

ベル(0x7)

¥b

バックスペース(0x8)

¥f

改頁 (0xc)

¥n

改行 (0xa)

¥r

キャリッジ・リターン (0xd)

¥t

タブ (0x9)

¥v

垂直タブ (0xb)

説明

¥<改行>空白

バックスラッシュと改行、さらにそれに続く全ての空白文字をスペース文字 1 個で置き換えます。このバックスラッシュシーケンスはコマンドを実際に解釈する前に処理が行われるという点で特殊です。つまりたとえ中括弧で囲まれた中にあっても置き換えが行われること、また置き換えたスペース文字は、中括弧やクォートの中になければワードの区切りになります。

¥¥

バックスラッシュ（``¥''）

¥ooo

ooo(1-3 文字の数字)でその文字の 8 進数での値を表します。

¥xhh

hh は 16 進数で、その文字の 16 進数での値を表します。数字は連続して指定可能です。

バックスラッシュ置き換えは中括弧で囲んだワードに対しては行いません。ただし上記のようにバックスラッシュ-改行置き換えだけは例外です。

[9] コマンドの最初のワードで、最初のキャラクタがくるべき場所にシャープ記号(``#'')があった場合は、そのシャープ記号およびそれに続く改行までの全ての文字列をコメントと判定します。このコメント文字はコマンドの先頭に現れた時だけ有効です。

[10] Tcl インタプリタは、それぞれの文字をコマンドのワードを作る際に 1 回だけ処理します。例えば変数置き換えが起こると、変数の値に対してそれ以上の置き換えは起こりません。変数値はそのままワードに入ります。コマンド置き換えの際にはネストしたコマンドを Tcl インタプリタの再帰的な呼び出しにより処理します。再帰的呼び出しの前には何の置き換えも行わず、ネストしたスクリプトの実行結果に対して更に置き換えが行われることもありません。

[11] 置き換えがコマンドのワードの区切りを変えることはありません。例えば変数置き換えの場合には、その値に空白文字を含んでいても変数の値が全体として一つのワードの一部になります。

使用例

―

1.55 tell

tell	オープンしているファイルの現在のアクセス位置を返します
形式	tell fileId
説明	
fileId の現在のアクセス位置を示す 10 進文字列を返します。fileId は open が返した値か、標準入出力のチャネルを表す stdin、stdout、stderr を指定してください。 注）WebTcl でネットワーク上のファイルをダウンロードする場合は、この機能は無効です。	
使用例	
－	

1.56 trace

trace	変数へのアクセスをトレースします
形式	trace variable name ops command trace vdelete name ops command trace vinfo name

説明

variable 形式の場合

このコマンドは、変数 name に対して ops で指定される方法のアクセスが起こった時に command で指定する Tcl コマンドを実行します。name には普通の変数の他に、配列の要素や配列全体も指定できます(name 小括弧で囲まれたインデックスなしのただの配列名も指定可能です)。

name が配列全体を示す時、その要素に対して操作が起これば command を実行します。

ops でどの操作をチェックするのかを指定します。指定できるのは次の文字の 1 文字以上の組合せです。

r

変数の値を参照した時に command を実行します。

w

変数の値をセットした時に command を実行します。

u

変数を消去した時に command を実行します。変数の消去は、unset コマンドを実行した時や、プロシジャの全てのローカル変数が、プロシジャからリターンする時です。インタプリタを消去した時にも変数を消去しますが、この場合すでに実行するためのインタプリタがなくなっているのでトレースを実行しません。

トレースが起こった時には、command に次の 3 つの引数を渡します。

command name1 name2 op

name1 と name2 にはアクセスした変数の名前が入ります。通常の変数の場合、name1 に変数名が入り、name2 は空文字列になります。配列変数の場合、name1 に配列名が入り、name2 には配列のインデックスが入ります。もしも配列全体を消去し、配列の特定の要素にではなく配列全体に対してトレースがかかっていた時には name1 に配列名、name2 には空文字列が入ります。op はその変数に対してどの操作が起こったのかを示し、前述の r、w、u のうちのどれかになります。

command はトレースを引き起こしたコードと同じコンテキストで実行します。プロシジャ内で変数をアクセスした時は、command はそのプロシジャと同じローカル変数にアクセスします。comannd を実行する時のコンテキストはトレースを設定した時のコンテキストとは通常異なります。command がプロシジャを起動した時にはそのプロシジャからトレース変数をアクセスするには upvar や uplevel を使用する必要があります。name1 は upvar で定義した変数をアクセスした場合、トレースをセットした変数の名前とは違う可能性があります。

説明

変数値の参照やセットに対するトレース時に、command でその変数を変更してトレース操作の結果を変えることができます。変数値の参照やセットに対するトレース時に command で変数値を変えると、その変えた値をトレース操作の結果として返します。command のリターン値は無視しますが、なんらかのエラーを返した時にはトレースのかかった操作は中止し、トレース用コマンドが出したエラーメッセージとともにエラーが返します。変数のセットに対するトレースでは、変数の値が変更した後で command を呼び出します。command 内でセットした値にオーバーライドして、別の値をセットすることも可能です。

変数値の参照やセットに対するトレースで command を実行している間はその変数に対するトレースは一時的にストップします。これは command によって行なわれた変数の読み書きは、再び command その他のトレースを呼び出すことなく直接行なわれるということです。しかし、command がその変数を消去した時には消去用のトレースを呼び出します。

変数の消去に関するトレースを呼び出した時には、その変数はすでに消去済みです。プロシジャからリターンする際の変数の消去では、トレースは(その一つ上で)リターンを待っているプロシジャの変数コンテキストで呼び出します。リターンしようとしているプロシジャのスタックフレームはすでに存在しません。変数消去のトレース時には、トレースが一時的にストップすることはありません。従ってトレースコマンドが新しいトレースを作りその変数にアクセスするとトレースを起動します。変数消去のトレース時のエラーはすべて無視します。

一つの変数に対して複数のトレースがかかっていると、一番最後に設定したトレースから順に呼び出します。あるトレースでエラーになると、その変数に対する残りのトレースの呼び出をキャンセルします。配列のある要素にトレースをかけていて、さらに配列全体に関してもトレースをかけている時には、その要素に対するトレースより先に配列全体のトレースを呼び出します。

vdelete 形式の場合

トレースは一度設定すると、trace vdelete コマンドで消去する、変数自体を消去するか、インタプリタが終了するまで有効です。配列のある要素を消去した時にはその要素に対するトレースは消去しますが、配列全体に対するトレースは消去しません。vdelete 形式では空文字列を返します。

vinfo 形式の場合

変数 name に現在セットしている各トレースの情報のリストを返します。リストの各要素は、そのトレースに関する ops と command です。name が存在しない時や、トレースが設定していない時は、空文字列を返します。

使用例

```
# $name の値の変数名に対して、変数の書き込みのトレースを設定する例
set psvwb(com) "trace variable $name w psvwb_Trace"
# $name の値の変数名に対して、トレースを削除する例
set psvwb(com) "trace vdelete $name w psvwb_Trace"
# $va の値の変数名に対して、トレース情報のリストを取得する例
set psvwb(temp) "trace vinfo $va"
```

1.57 unknown

unknown	存在しないコマンドを使用した時のハンドラ
形式	unknown cmdName arg [arg ...]
説明	
このコマンド自体は Tcl の内部コマンドではありません。このコマンド自体が unknown という名前のコマンドを定義することになります。Tcl インタプリタが、定義していないコマンド名を検出すると、Tcl はまず unknown という名前のコマンドがないかチェックします。なければインタプリタはエラーを返します。unknown コマンドが存在する場合、元の存在しないコマンドに対するコマンド名と引数に対して置き換えを行い、それを引数として unknown コマンドを呼び出します。unknown コマンドは普通 cmdName という名前のプロシジャをライブラリディレクトリから探したりして、短縮形で書いたコマンドを元の名前に戻したり、あるいは自動的にそのコマンドを外部コマンドのプロセスとして実行する目的で使用されます。unknown コマンドの結果は元の存在しないコマンドの結果として代わりに使用します。	
使用例	
―	

1.58 unset

unset	変数を消去します
形式	unset name [name name ...]

説明

このコマンドは一つ以上の変数を消去します。name は変数名を指定します。name が配列の要素を指定する時は、配列の他の要素には影響せず、その要素だけを消去します。name が小括弧で括っていない単なる配列名の場合は、その配列全体を消去します。unset コマンドは空文字列を返します。指定した変数の中に一つでも存在しない変数名があればエラーが発生します。またエラーの変数名の後に指定した変数の消去は実行しません。

使用例

```
# gsamp001.psv から抜粋。管理用の配列変数 gs001 を配列ごと消去する例
unset gs001
# st_op.psv から抜粋。図形を元の色に戻したあと、保存しておいた元の色を消去する例
foreach i $stval($stid,zuobjs) {
        g_color_id $i $stval(cid$i)
        unset stval(cid$i)
}
```

1.59 uplevel

uplevel	スクリプトを別のスタックフレームで実行します
形式	uplevel [level] arg [arg ...]

説明

全ての引数 arg を、上位スタックフレームで対応する変数のコンテキストで実行します。uplevel はその実行の結果を返します。

level が整数値の時は、プロシジャのスタックを相対的にどれだけ移動するかを示します。level が``#数字'' という形であればその数字はスタックフレームのレベルの絶対番号を示します。level 省略時のデフォルト値は 1 です。最初の引数が数字や#ではじまる時には level の省略はできません。

たとえば、トップレベルからプロシジャ a を呼び出して、そこから b を呼出し、さらに b が c を呼び出しているケースを想定してください。プロシジャ c で uplevel コマンドを実行した場合、level が 1 あるいは#2 を指定した時にはコマンドをプロシジャ b の変数コンテキストで実行します。level が 2 か#1 を指定した時は、コマンドをプロシジャ a の変数のコンテキストで実行します。level が 3 または#0 を指定した時は、コマンドをトップレベルで実行します（この場合グローバル変数だけがアクセス可能です。）。

uplevel コマンドは、実行中は呼び出したプロシジャをプロシジャのスタックから一旦消します。

前述の例で、プロシジャ c が

```
uplevel 1 {set x 43; d}
```

というコマンドを実行したと仮定します。ただし d もまた別のプロシジャです。set コマンドはプロシジャ b のコンテキスト中の変数 x の値を変更し、プロシジャ d をプロシジャ b から呼び出されたとして、レベル 3 で実行します。

もしプロシジャ d が

```
uplevel {set x 42}
```

というコマンドを実行したとすると、set コマンドはプロシジャ b のコンテキストの前述の変数 x の値を変更することになります。つまり、プロシジャ d を実行中はプロシジャ c はコールスタックに無いように見えます。

※スタックフレームのレベルは、コマンド ``info level'' で知ることができます。

uplevel を応用すると、スタックフレームを跨った制御文のような処理が実装可能です。

使用例

－

1.60 upvar

upvar	別のスタックフレームにある変数にリンクを張ります
形式	upvar ?level? otherVar myVar ?otherVar myVar ...?

説明

このコマンドでは、カレントプロシジャのローカル変数を、呼び出した上位のプロシジャの変数やグローバル変数とリンクして直接アクセス可能になります。level の表記は uplevel コマンドの Level の仕様を参照してください。また、最初の otherVar が#や数字ではじまっていなければ、level は省略可能です(デフォルトは 1)。upvar は引数 otherVar のそれぞれについて、level で指定したプロシジャのスタックフレーム(または level が#0 ならグローバルレベル)にある otherVar という変数を、カレントプロシジャから myVar という名前でアクセスできるようにします。この場合コマンドを呼び出した時点で変数 otherVar が存在する必要があります。myVar は最初に参照した時に作成します。upvar はプロシジャの中だけで使えます。myVar も配列の要素は指定できません。ただし、otherVar は配列の要素を指定可能です。upvar は空文字列を返します。

以下のように、upvar コマンドを使うと C++言語の引数の参照渡しと同じ処理が実装可能です。

```
proc add2 name {
  upvar $name x
  set x [expr $x+2]
}
```

add2 は引数として変数の名前をとり、リンクを張った変数 x の値に 2 を加えます。これはリンクを張った上位スタックフレームの変数も同時に値を設定します。

upvar した変数(例えば、上記の例では add2 の変数 x)を unset した時は、unset の対象はリンクを張った先の変数ではなく、リンク元の変数 x のほうを消去します。

使用例

```
# gsamp023.psv から抜粋。サンプルの配列変数名をわたし、配列自体にリンクを貼り、要素にアクセスする例
# 呼び出し側の例
gs023p gs023 u 50.0
# 呼出される側の例
proc gs023p {gs023 mode add} {
        upvar $gs023 gs
        scan [g_ortho $gs(gc)] "%f %f %f %f" x1 y1 x2 y2
```

→$gs(gc)は呼び出し側の$gs023(gc)とリンクしているので、上位で保存する g_context の戻り値を参照可能

1.61 while

while	while ループコマンド
形式	while test body

説明

while コマンドは test を式として評価します。式の値は適切な論理値になる必要があります。式の値が真の場合は、body を Tcl インタプリタで実行します。body を一度実行すると、test を再び評価します。この繰り返しは、test の評価結果が偽になるまで行なわれます。body の中で continue コマンドを実行すると、それ以後の body の実行を中断し、test の再評価に移行します。break コマンドを実行すると、while コマンドの実行を直ちに終了します。while コマンドは常に空文字列を返します。

使用例

```
# color.psv から抜粋。ファイルを行単位で入力して EOF に達するまでループする処理
while { ![eof $fd] } {
        if { [catch { gets $fd macro } str] != 0 } {
                close $fd
                return -1
        }
        scan $macro "%s %s %s %s" dm1 dm2 no rgb
        …
}
```

2 数学関数

expr コマンド内で使用可能な、数学関数を説明します。

2.1 abs

abs	絶対値の取得
形式	abs(x)
引数	任意の数値を指定可能です。
戻り値	x の絶対値を返します。

2.2 acos

acos	逆余弦値の取得
形式	acos(x)
引数	逆余弦値を計算する値。-1 ～1 までの範囲の値が指定可能です。
戻り値	x の逆余弦値を返します。戻り値：戻り値は 0～π ラジアンの範囲の値となります。

2.3 asin

asin	逆正弦値の取得
形式	asin(x)
引数	逆正弦値を計算する値。-1～1 の範囲の値が指定可能です。
戻り値	x の逆正弦値を返します。戻り値は、-π/2 ラジアンからπ/2 ラジアンの範囲の値となります。

2.4 atan

atan	逆正接値の取得
形式	atan(x)
引数	逆正接を計算する値。任意の数値を指定可能です。
戻り値	x の逆正接値を返します。x が 0 の場合は 0 を返します。戻り値は、-π/2 ラジアンからπ/2 ラジアンの範囲の値となります。

2.5 atan2

atan2	y/x の逆正接値の取得
形式	atan2(x,y)
引数	y/x の逆正接を計算します。任意の数値を指定可能です。両方の引数の符号を使って戻り値の象限を定めます。
戻り値	y/x の逆正接値を返します。2 つの引数が 0 の場合は 0 を返します。戻り値は、-πラジアンからπラジアンの範囲の値となります。

2.6 ceil

ceil	浮動小数点値の切り捨て
形式	ceil(x)
引数	浮動小数点の値を指定します。
戻り値	x を下回らない最小の整数値を返します。

2.7 cos

cos	余弦値の取得
形式	cos(x)
引数	ラジアン単位の角度を指定します。
戻り値	x の余弦値を返します。

2.8 cosh

cosh	双曲線余弦値の取得
形式	cosh(x)
引数	ラジアン単位の角度を指定します。
戻り値	x の双曲線余弦値を返します。

2.9 double

double	整数値の浮動小数点化
形式	double(x)
引数	整数値を指定します。
戻り値	x を浮動小数点化して返します。

2.10 exp

exp	指数の取得
形式	exp(x)
引数	任意の値を指定可能です。
戻り値	x の指数を返します。

2.11 floor

floor	浮動小数点値の切り上げ
形式	floor(x)
引数	浮動小数点の値を指定します。
戻り値	x を越えない最大の整数値を返します。

2.12 fmod

fmod	浮動小数点値の剰余演算値の取得
形式	fmod(x,y)
引数	x,y ともに浮動小数点値を指定します。
戻り値	x を y で割った時の剰余演算値を返します。

2.13 hypot

hypot	hypot の取得
形式	hypot(x,y)
引数	任意の値を指定可能です。
戻り値	x*x + y*y の平方根を返します。

2.14 int

int	整数値の取得
形式	int(x)
引数	浮動小数点の値を指定します。
戻り値	x を整数化して返します。

2.15 log

log	自然対数の取得
形式	log(x)
引数	対数計算の対象となる値を指定します。
戻り値	x の自然対数を返します。

2.16 log10

log10	常用対数の取得
形式	log10(x)
引数	対数計算の対象となる値を指定します。
戻り値	x の常用対数(底=10)を返します。

2.17 pow

pow	ｙ乗の取得
形式	pow(x,y)
引数	x は底、y は指数を指定します。
戻り値	x を y 乗した値を返します。

2.18 rand

rand	乱数の取得
形式	rand()
引数	なし
戻り値	0～1 の間の乱数を返します。

2.19 round

round	浮動小数点値の丸め
形式	round(x)
引数	浮動小数点の値を指定します。
戻り値	x に最も近い int 値に丸めた値を返します。

2.20 sin

sin	正弦値の取得
形式	sin(x)
引数	ラジアン単位の角度を指定します。
戻り値	x の正弦値を返します。

2.21 sinh

sinh	双曲線正弦値の取得
形式	sinh(x)
引数	ラジアン単位の角度を指定します。
戻り値	x の双曲線正弦値を返します。

2.22 sqrt

sqrt	平方根の取得
形式	sqrt(x)
引数	負でない浮動小数点値を指定します。
戻り値	x の平方根を返します。

2.23 srand

srand	乱数発生器の初期化
形式	srand(x)
引数	整数を指定します。
戻り値	整数 x を乱数の種として、乱数発生器を初期化します。

2.24 tan

tan	正接値の取得
形式	tan(x)
引数	ラジアン単位の角度を指定します。
戻り値	x の正接値を返します。

2.25 tanh

tanh	双曲線正接値の取得
形式	tanh(x)
引数	ラジアン単位の角度を指定します。
戻り値	x の双曲線正接値を返します。