LScript
日本語リファレンスマニュアル

著者： Brian Marshall、Scott Wheeler

日本語訳・マニュアル制作：株式会社ディ・ストーム

Manual version: 1.0J - beta

August, 2002

LScript 日本語リファレンスマニュアル 目次

第４章：OBJECT AGENTS リファレンス

第５章：共通データメンバとメソッド

第６章： MESH OBJECT AGENTS

第16章：CONTROL OBJECT AGENTS

第17章：SHADER OBJECT AGENTS

第１８章： SURFACE OBJECT AGENTS

第１９章： TEXTURE LAYER OBJECT AGENTS

第２０章： TEXTURE OBJECT AGENTS

第２１章： IMAGE FILTER OBJECT AGENTS

第２４章： PARTICLE OBJECT AGENTS

第２５章： MOTION OBJECT AGENTS

第２６章： CHANNEL OBJECT AGENTS

マニュアルの表記方法について

書体と色

このマニュアルは、LightWave 3D [7] 日本語マニュアルの表記方法に準じて記述されていますが、プログラミングの解説という性質上、以下の違いがあります。

機能や用語

LScript や LightWave 3D の機能、メニュー名、および一般的なプログラミング用語が初めて出てくる箇所は、**黒の太字**で書かれています。

例：

このような場合、**変数**を使えばスクリプト実行中に、このデータを…

コマンドや構文

コマンドや構文は、青で書かれています。

例：

```
main
{
  // このスクリプトは info ボックスを画面上に表示します
  // 変数テキストの値は "Hello World!"です
  …
}
```

コメント行

本文のスクリプト中に、"//"や"/*"で囲まれたコメント行があります。このマニュアルでは、分かりやすく解説するためにコメント行を日本語で記述していますが、実際の LScript では英数字しか使用できません。

第 1 章：共通コマンド

この章では、レイアウトとモデラーの LScript 双方で使用できるコマンドについて解説していきます。数学関数や変数変換、文字列操作、システム情報、ファイル入出力におけるコマンドと関数について解説します。インターフェイスコントロールについては次章で解説します。

center

`center` コマンドは、三次元空間内における指定した二点の中心点の座標値を返します。このコマンドでは様々なパラメータ数とタイプを受け取ります。配列を受け取る場合は、最初の二つのエレメントが二つのベクトル（定数または変数）を持ち、そうでない場合には六つの数値を受け取ります。この六つの数値のうち最初の三つが一番目のベクトルを、残りの三つが二番目のベクトルを表します。`center()` はベクトルデータタイプを返します。

```
// boundingbox() はベクトル値を二つ返しますので
// そのまま出力値を center()へ渡します
objectCntr = center(boundingbox());
```

extent

`extent` コマンドは `center()` と同数/同種類の引数を受け取りますが、`center()` では一組のベクトルの中心点を計算するのに対し、このコマンドは二点間の範囲（距離）を返します。`extent()` は各軸に対する範囲を表す要素（x,y,z）を持つベクトルデータタイプの値を返します。

```
objectSize = extent(boundingbox());
```

angle

`angle` 関数は、指定した平面上（XZ、YZ または XY）にある二点間の角度を度数で返します。引数は二点を表す二つのベクトル、第 3 引数として角度計算を行う平面に対しての垂直軸を指定します。

```
...
editbegin();
  pnt1 = pointinfo(points[1]);
  pnt2 = pointinfo(points[2]);
editend();
// YZ 平面における角度を返します
info(angle(pnt1,pnt2,X), degrees" );
...
```

regexp

regexp()関数を使用すると、LScriptは文字列から正規表現を構築することが出来ます。この関数から正規表現パターンを入力します。コンパイル用に二つの個別の引数が提供されていれば、正規表現に置き換えることが可能です。

```
// ファイル内に存在する"PUB"で始まる行を表示します
// 使用される正規表現をコンパイルします
expr = regexp( "^PUB" );
// regexp()関数は正規表現のコンパイルに失敗すれば'nil'を返します
if(expr == nil)
  return;
if((file = File( "nfa.c" ," r" )) == nil)
  return;
while(!file.eof())
{
  line = file.read();
  if(line == expr)
  info(line);
}
```

spawn

spawn コマンドは外部プロセスの実行を開始します。通常、外部プロセスはこのコマンド内部において実行します。この場合、spawn()コマンドは指定したプロセスの実行結果コードを返します。

ただしスクリプトに@asyncspawn プラグマ指示文が宣言されている場合、spawn()コマンドは外部プロセスの起動に成功すればプロセスIDを返します。プロセスIDはwait()やterminate()など他のプロセス制御関数への引数として必要になります。

```
// ScreametNet II を同時に起動
snID = spawn( "lwsn -2  ",
getenv( "TMP" )," \\snii.job  ",
getenv( "TMP" )," \\snii.rpl" );
if(snID == nil)
{
  error( "Cannot create ScreamerNet process" );
  ...
```

wait

wait コマンドはspawn()に対する同期関数として提供されます。プロセスID（spawn()関数から返されるID）が引数として渡されると、外部プロセスが終了するまでスクリプト実行を停止状態にし、プロセスの終了状態を返します。

terminate

`terminate()`コマンドは、プロセス制御コマンドのもう一つの関数です。`terminate()`はWindowsに特定のプロセスIDを終了させます。

```
terminate(snID); // ScreamerNet を終了させます
```

sleep

`sleep()`コマンドを使うと、スクリプトは指定したミリ秒間一時停止状態にし、時間を過ぎるとスクリプト（そしてモデラー）のWindowsタイムスライスを開放します。1000ミリ秒は一秒です。

```
sleep(1000); // 1秒間 sleep をかけます
```

変換

変換コマンドは変数をある一つの型から他の型へと変換します。これらコマンドには同意義のメソッドがいくつか存在します。詳しくは、このリファレンスマニュアルの「Variable Object Agents」の章を参照してください。

number

`number()`は文字列引数を受け取り、数値（浮動小数点数）へと変換します。

```
str = "69.6";
t = number(str); // 69.9 を返します
t = number("The answer is: .1593");
// .1593 を返します
```

integer

`integer()`は文字列引数を受け取り、整数値を返します。文字列に小数点が含まれていれば取り除きます。

```
str = "69.6";
t = integer(str); // 69 を返します
t = integer(round(number("69.9"))); // 70 を返します。
```

vector

`vector`コマンドは文字列引数を受け取り、ベクトルへと変換します。

```
str = "4 6.5 8";
t = vector(str); // <4,6.5,8> を返します
```

string

string()コマンドは可変数の引数を受け取ります。それぞれの引数はサポートされているLScriptのデータタイプであれば何を指定しても構いません（配列以外）。この引数を基に文字列を構成します。

```
t = string( "Value  ",val," exceeds max limit of  ", 10 * items," items" );
```

ascii

ascii()変換関数は、引数として文字列を受け取り、それに対応する整数値を返します。印字抑止文字などを比較する際に便利な関数です。

```
if(ascii(line[x]) ==  '\t' )
```

hex

hex変換ツールは数値を16進数へと変換します。

```
obj = getfirstitem(LIGHT);
info(hex(obj.id)); // 表示 "0x20000000"
```

octal

octal変換ツールは数値を8進数へと変換します。

```
value = octal(1.7); // 結果:  "\1"
```

ファイル管理

ファイル管理コマンドはファイルやディレクトリを扱います。

File

`File()`コンストラクタはファイルのディスク上の位置を示す文字列と、ファイルを開くときの状態を示すオプションモード文字列を受け取ります。ファイルコマンドは**File Object Agent**を返します。この**Objects　Agents**やその属性についての解説は、このリファレンスマニュアルの「File Objects Agents」の章を参照してください。

```
sceneFile = File( "/temp.lws" ," w" );
```

以下に`File`オープンモードのリストを紹介します。これらのモードはC言語で提供されるものと同じモードです。

```
r read (テキスト)
w write (テキスト)
a append (テキスト)
b binary ("rb" は読取専用バイナリ)
+ with update (ファイルポインタは任意の場所に移動可能)
```

例えばランダムバイナリ読み出しアクセスでファイルを開く場合、オープンモードには"`rb+`"と指定します。オプションのオープンモードを指定しない場合には、ファイルはデフォルトであるアスキー読み出しモード（`r`）で開かれます。

filerename

`filerename()`コマンドは、ディスク上のファイルのリネームを行います。第1引数には既存のファイル名を、第2引数には適用したい新規ファイル名称を指定します。

```
filerename( "c:\\test.txt" ,  "C:\\test2.txt" );
```

filestat

`filestat()`関数は、ディスク上のファイルに割り当てられている様々なシステム関連の値を返します。`filestat()`関数に適切なファイル名称が渡されると、ファイルのアクセス日時、作成日時、更新日時ファイルサイズ、ファイルへのリンク数（UNIXまたはNTFSファイルシステムのみ）、ファイルオーナーユーザーID（UNIXのみ）、ファイルのグループID（UNIXのみ）などの情報が順に返されます。

```
(a,c,m,s,l,u,g) = filestat( "/etc/profile" );
```

値は全て整数値であり、アクセス、作成、修正日時は`time()`や`date()`関数用の引数として使用できます。

fullpath

fullpath()関数は文字列の値（パス）を一つ受け取ります。渡されたパスが相対パスであればフルパスを返し、既にフルパスの場合にはそのままの値が返されます。

```
path1 = fullpath( "test2.txt" );
// 結果: "E:\content\test2.txt"
path2 = fullpath( "c:\\test2.txt" );
// 結果: " c:\test2.txt"
```

getdir

getdir()コマンドは、以下の定数（文字列定数/定数値）に対しアプリケーションによって使用される作業値を返します。

"Fonts" /FONTSDIR モデラー
"Macros" /MACROSDIR モデラー
"Install" /INSTALLDIR レイアウト/モデラー
(LWのインストールディレクトリ)
"Objects" /OBJECTSDIR レイアウト/モデラー
"Images" /IMAGESDIR レイアウト/モデラー
"Motions" /MOTIONSDIR レイアウト/モデラー
"Temp" /TEMPDIR レイアウト/モデラー
"Plugins" /PLUGINSDIR レイアウト/モデラー
"Settings" /SETTINSDIR レイアウト/モデラー
(もしくはConfigs)
"Content" /CONTENTDIR レイアウト
"Scenes" /SCENESDIR レイアウト
"Hierarchies" /HIERARCHIESDIR レイアウト
"Surfaces" /SURFACESDIR レイアウト
"Output" /OUTPUTDIR レイアウト
"Animations" /ANIMATIONSDIR レイアウト
"Envelopes" /ENVELOPESDIR レイアウト
"Previews" /PREVIEWSDIR レイアウト
"Command" /COMMANDDIR レイアウト

以下に記す形式においてこれらディレクトリを取得できます。

```
install = getdir( "Install" );
plugins = getdir(PLUGINSDIR);
```

getfile

getfile()はファイル選択のインターフェイスを提供します。ユーザにディスク上のファイルを指定させる必要が生じた場合、getfile()はファイル選択ダイアログボックスを出し、ユーザーが選択したファイル名称（パスも含む）を返します。キャンセルボタンが押されると'nil'が返ります。

getfile()のパラメータは全てオプションです。ダイアログボックスのタイトル、ファイル名称のマスク、デフォルトディレクトリなどが指定可能です。

```
getfile([title // 文字列： ダイアログボックスのタイトル
    [,mask // 文字列： 使用するファイルマスク
        // (ワイルドカード可)
    [,dir // 文字列： ダイアログが開いた段階での
      // ディレクトリパス
    [,reqType]]]]
      // ブール値： 0 = 保存, 1 = 開く
```

matchfiles

matchfiles()コマンドはディレクトリをスキャンし、パターンと一致するファイルやフォルダを調べていきます。matchfiles()からは指定したパスと検索パラメータに一致するファイルが入った配列を返します。

```
path1 = matchfiles( "c:\\" ,  "*.txt" );
```

matchdirs

matchdirs()コマンドは、指定した検索パラメータに一致するフォルダの配列を返します。

```
path1 = matchdirs( "c:\\" ,  "*.*" );
```

Unix 形式コマンド

chdir

chdir()はLScriptが提供するディレクトリ管理用の三つの関数のうちの一つです。この関数を使用する時には、スクリプトが開始するディレクトリから"作業"ディレクトリを変更することが可能です。またこのコマンドからは現在の"作業"ディレクトリをフルパスを表す文字列が返されます。

chdir()コマンドは、特定のディレクトリ内から実行させなくてはならない外部プロセスを起動する場合に便利です。ScreamerNetはこのプロセスの良い例です。コンフィグファイルや画像等のサポートファイルを適切に位置づけるため、ScreamerNetはLightWave￥Programsディレクトリ内部から起動しなければなりませんが、LScriptは既に全く異なる場所で実行されています。以下のコードは断片ですが、chdir()関数がどのようにしてこの特殊な状況を解決しているのか紹介します。

```
// "作業"ディレクトリをScreamerNetによって要求される
// ディレクトリへと変更します

if((snDir = getenv( "LW3D" )) == nil)
  // 環境は適切ですか？
  oldDir = chdir( "\\windows\\apps\\newtek\\programs" );
else
  oldDir = chdir(snDir);

snID = spawn( "lwsn -2 ", getenv( "TMP" )," \\snii.job ", _getenv( "TMP" )," \\snii.rpl" );
chdir(olddir);
// 起動時のディレクトリへと戻ります
```

chdir()関数が作業ディレクトリをある特定のディレクトリへと設定するのに失敗した場合には'nil'が返されます。

mkdir

mkdir()はディレクトリ管理に関するもう一つの関数です。LScriptはmkdirを使用して新規ファイルシステムディレクトリを作成することが出来ます。この関数はディレクトリを表す（オプションでパス付き）文字列を受け取り、結果コードを返します（0はエラー無し）。

```
mdir( "c:\\temp2" );
```

rmdir

rmdir()は既存のディレクトリを削除します。このコマンドが成功すると削除されるディレクトリは完全に空になります。mkdir()と同様、このコマンドはディレクトリ（オプションでパス付き）を指定する文字列を受け取り、結果コードを返します。

```
rmdir( "c:\\temp2" );
```

数学コマンド

数学コマンドには数学関数で使用されるコマンドが含まれています。

sqrt

sqrt()関数は平方根を計算します。引数として数値が渡され、その値の平方根を返します。

```
value = sqrt(4); // 結果: 2
```

exp

exp()関数は指数を計算します。引数として数値が渡され、その値の指数を返します。

```
value = exp(2); //結果: 7.38906
```

log

log()関数は自然対数を返します。引数として数値を渡し、その値の自然対数を返します。

```
value = log(124.2); //結果: 4.82189
```

sin

sin()関数はラジアンで指定された角度の正弦を返します。引数としてラジアン角度を受け取り、その角度の正弦値を返します。

```
value = sin(241.8); //結果: 0.102454
```

cos

cos()関数はラジアンで指定された角度の余弦を返します。引数としてラジアン角度を受け取り、その角度の余弦値を返します。

```
value = cos(153.2); //結果: -0.739789
```

tan

tan()関数はラジアンで指定された角度の正接を返します。引数としてラジアン角度を受け取り、その角度の正接値を返します。

```
value = tan(135.0); //結果: -0.0887158
```

asin

asin()関数はラジアンで指定された角度の逆正弦を返します。引数としてラジアン角度を受け取り、その角度の逆正弦値を返します。

```
value = asin(.2); //結果: 0.201358
```

acos

acos()関数はラジアンで指定された角度の逆余弦を返します。引数としてラジアン角度を受け取り、その角度の逆余弦値を返します。

```
value = acos(.53); //結果: 1.0122
```

atan

atan()関数はラジアンで指定された角度の逆正接を返します。引数としてラジアン角度を受け取り、その角度の逆正接値を返します。

```
value = atan(129.2); //結果: 1.56306
```

cosh

cosh()関数はcos()関数の双曲線値を計算します。

```
value = cosh(82.2); //結果: 1.55863
```

sinh

sinh()関数はsin()関数の双曲線値を計算します。

```
value = sinh(.13); //結果: 0.130366
```

tanh

tanh()関数はtan()関数の双曲線値を計算します。

```
value = tanh(.01); //結果: 0.00999967
```

cot

cot()関数はラジアンで指定された角度の余接を返します。引数としてラジアン角度を受け取り、その角度の余接値を返します。

```
value = cot(21); //結果: -0.654665
```

csc

csc()関数はラジアンで指定された角度の余割を返します。引数としてラジアン角度を受け取り、その角度の余割値を返します。

```
value = csc(121.3); //結果: 1.06403
```

abs

abs()関数は数値の絶対値を返します。数値の符合に関わらず返される値は常に正の値です。

```
value = abs(-121.2); //結果: 121.2
```

ceil

ceil()関数は引数として渡された数値以上で最も小さい整数値を返します。

```
value = ceil(-1.201); //結果: -1
```

floor

floor()関数は引数として渡された数値以下で最も大きい整数値を返します。

```
value = floor(-1.201); //結果: -2
```

deg

deg()関数は単一の角度パラメータ（ラジアン単位）を受け取り、度数に変換します。

```
value = deg(241.3); //結果: 13824.5
```

random

random()関数は擬似乱数整数値を生成します。この関数は二つの整数値または浮動小数点数値を受け取ります。整数を指定した場合には二つの値の範囲内にある乱数整数値を、浮動小数点数を指定した場合には浮動小数点数値を返します。値は正負どちらでも可能です。

```
triggerFrameInt = random(20,50);
triggerFrameFloat = random(20.2, 50.2);
```

min

min()関数は二つの引数値のうち、小さい方の値を返します。

```
t = min(1,7); //結果: 1
```

max

max()関数は二つの引数値のうち、大きい方の値を返します。

```
t = max(1,7); //結果: 7
```

mod

mod()関数は第 1 引数を第 2 引数で割ったときの剰余を返します。また LScript の命令子"%"も同義の命令子です。

```
t = mod(34.5,6); // 4.5 を返します
もしくは
t = 34.5 % 6;
```

pow

pow()関数は第1引数を第2引数で累乗した値を返します。

```
t = pow(5,3); // 125 を返します
```

hypot

hypot()関数は第1引数と第2引数に対応する辺を持つ直角の斜辺を計算します。

```
t = hypot(3,5);
```

rad

rad()関数は度数で指定される単一の角度パラメータを受け取り、ラジアン値に変換します。

```
t = rad(degrees);
// 次の動作と等しい: degrees * (PI / 180)
```

randu

randu()関数は範囲0～1にある乱数の値を返します。

```
rad = 0.05 * maxlen * (1 + randu());
```

range

range()関数は、第1引数が第2引数と第3引数で指定された値の範囲内に包括されているかどうかを示すブール値（true もしくは false）を返します。

```
if(range(top,3,8))
```

selector

selector()関数は、はじめの二つのパラメータを比較し、第1引数が第2引数よりも小さければ第3引数を返します。第1引数が第2引数よりも大きければ第4引数を返します。

```
t = selector(4,7,15,3); // 15 を返します
t = selector(15,7,7,10); // 10 を返します
```

step

step()関数は、二つの引数を評価し、第2引数が第1引数よりも小さければ0を、第2引数が第1引数よりも大きければ第3引数を返します。

```
t += step(y,10,.25); // .25 によりステップ
```

round

round()関数は、第2引数で指定した小数点以下の桁数で第1引数の値を丸めます。第2引数の値が0であれば、数値は最も近い整数値へと丸められます。

```
t = round(37.7,0); // 38 を返します
t = round(4.4394,2); // 4.44 を返します
```

frac

frac()関数は、数値の小数部分を返します。

```
t = frac(46.75); // 0.75 を返します
```

fac

fac()コマンドは指定された数値パラメータの階乗値を計算します。

```
t = fac(5); // 120 を返します
```

vmag

vmag()関数は三次元ベクトルの絶対値を計算します。

```
t = vmag(x,y,z);
// 位置の絶対値を計算します。
```

空間内における任意のベクトル間の距離を減算し、vmag()関数に渡すだけで、簡単に距離を計算することが出来ます。

```
v1 = <1,3,5.4>;
v2 = <.054,2,90>;
t = vmag(v1 - v2);
```

cross2d

cross2d()は二つの浮動小数点数値の外積を計算します。

```
value = cross2d(21.4, 53.2, 34.2, 12.1);
//結果: -1560.5
```

cross3d

cross3d()は二つの三次元ベクトルの外積を計算します。

```
a = <13.2, 2.5, 5.2>;
b = <122.1, 23.4, 53.2>;
value = cross3d(a,b);
//結果: <11.32, -67.32, 3.63>
...
```

dot2d

dot2d()は二つの浮動小数点数値の内積を計算します。

```
value = dot2d(112.1, 24.8, 63.3, -125.6);
//結果:10210.8
```

dot3d

dot3d()は二つの三次元ベクトルの内積を計算します。

```
value = dot3d(<12.2, 42.5, -12.3>, <125.5, 32.6, -23.1>);
//結果: 3200.73
```

sec

sec()関数は正割値を計算します。

```
value = sec(123.3); //結果: -1.40371
```

gamma

gamma()関数は値の階乗関数を補間します。

```
value = gamma(121.7); //結果: 121.7
```

normalize

normalize()関数はベクトル値の引数を受け取り、正規化されたベクトルを返します。

```
vec = 123.2;
vec = normalize(vec);
//結果: <0.992815, -0.0257874, 0.116849>
...
```

文字列

split

split()はパス/名称キャラクタ文字列に対して特定の動作を行います。文字列を受け取ると、引数を分割してドライブ、パス、ファイル名称、ファイル名拡張子を表す四つの要素をもつ配列を返します。

```
file = getfile( "Select A Scene..." ," *.lws" );
if(file == nil)
  return;
if(fileexists(file))
{
  base = split(file);
  // コンポーネントを取得します(base は配列となります)
  ...
```

存在しない文字列コンポーネント（例えばドライブ名称が存在しない等）には'nil'が返されます。

strleft

strleft()関数は、文字列の左端からの部分文字列を返します。この関数は二つのパラメータ値を受け取ります。第1引数は処理される文字列、第2引数は抽出する文字数を指定します。

```
value = strleft( "This string." ,3);
//結果:  "Thi"
```

strright

strright()関数は文字列の右端からの部分文字列を返します。この関数は二つのパラメータ値を受け取ります。第1引数は処理される文字列、第2引数は抽出する文字数を指定します。

```
value = strright( "This string." ,3);
... //結果:  "ng."
```

strsub

strsub()は文字列抽出関数です。この関数ではどちらか端からではなく、任意の位置から始まる文字列を任意の文字数分だけ抽出できます。

```
t = strsub( "Ted Philmore" ,3,3); // "d P" が返されます
```

strupper

strupper()関数は指定された文字列を大文字にして返します。

```
value = strupper( "upper-case" ); //
結果: UPPER-CASE
```

strlower

strlower()関数は指定された文字列を小文字にして返します。

```
value = strupper( "LOWER-CASE" ); //
結果: lower-case
```

parse

parse()は二つの引数を受け取ります。第1引数に予想トークン文字列、第2引数には処理する文字列を渡します。この関数は文字列（第2引数）を構成しているトークンの配列を返します。

```
str =  "23,45,69.6,100" ;
tokens = parse( "," ,str);
//  "23" ,  "45" ,  "69.9"  と  "100" を返します
```

format

format()関数は文字列内のトークンを指定した値で置換することにより文字列をフォーマットします。C言語のprintf()関数とよく似た関数です。個別のデータエレメントに対しても使用可能ですが、配列向きの関数です。

```
format(<template>,<data>[,<data>...])
```

トークンはドル記号の後に続く、引数で指定された数値インデックスで指定されます。インデックス値は連続した値である必要はなく、また全てのエレメントにアクセスする必要もありません。

引数にはそれぞれ単一のデータタイプ（変数もしくは定数）であるか、または置換に使用されるための単一配列を指定しても構いませんが、両方を指定することは出来ません。関数は全ての置換処理を施された文字列を返します。

```
d = date();
info(format( "Today is $7, $6 $1, $3" ,d));
(h,m) = time();
ap =  "AM" ;
if(h > 11)
{
  ap =  "PM" ;
  h -= 12 if h > 12;
}
h = 12 if h == 0;
info(format( "Time is currently $1:$2"  + ap,h,m));
```

size

size()コマンドは特定のデータタイプの変数のサイズを確定します。数値であれば1を、ベクトルであれば3を返します。文字列引数は文字列内部の文字数を、配列引数の場合には配列内の実際のエレメント数（配列全体のサイズではなく）を返します。

```
str = "23,45,69.6,100";
tokens[10] = nil;
t = size(tokens); // 0 を返します
tokens = parse(",",str); // 要素 4 の配列を返します
t = size(tokens); // 4 を返します
tokens[8] = 1.0;
t = size(tokens); // 5 を返します
```

sizeof

sizeof()関数は配列のサイズを返します。

```
tmp[1] = 1;
tmp[2] = 2;
tmp[3] = 3;
info(sizeof(tmp));
```

store

store コマンドを使うとスクリプトは何度起動を繰り返しても変数の値を保持しておくことが出来ます。store()コマンドは二つの引数を受け取ります。第 1 引数は保存する値を一意に認識するための文字列定数（または文字列が入った変数）、第 2 引数は実際に保存させる値です。

```
store("myName",myName);
...
```

recall

recall コマンドを使うと、スクリプトは何度起動を繰り返しても変数の値を保持しておくことが出来ます。recall()は前回 store()で保存しておいた値を取り出します。store()と同様、二つの引数を受け取ります。第 1 引数は保存する値を一意に認識するための文字列定数（または文字列が入った変数）、第 2 引数はこの識別子に対し store()が発行されていない場合に返す初期値です。

```
heading = recall("heading",15);
```

globalstore

globalstore()関数は store()コマンドと同じような働きをしますが、この関数はスクリプトが異なっても同一構造を持ちさえすれば機能します。

```
globalstore("heading", 23);
```

globalrecall

globalrecall()関数は recall()コマンドと同じような働きをしますが、この関数はスクリプトが異なっても同一構造を持ちさえすれば機能します。

```
heading = globalrecall("heading", 8);
```

システム

time

time()関数は現在時刻を時、分、秒、チックという複数のアイテムとして返します。時刻値が引数として提供されると（1900年1月1日からの経過秒数として定義)、戻り値はタイムインデックスへの相対値となります。時は24時間制なので範囲は0～23です。分の範囲は0～59です。チック値は1900年1月1日からの経過秒数です。

```
(h,m,s,t) = time();
```

date

date()関数は日、月、年（4桁)、曜日、年間通算日（ユリウス暦)、現在の月を表す文字、現在の曜日を表す文字を複数アイテムとして返します。タイムインデックス値が引数として提供された場合、date()はタイムインデックス値からの相対値を返します。曜日は日曜日を1として始まり、年間通算日は1～365の範囲となります。

```
(d,m,y,w,j,sm,sw) = date();
```

getenv

C言語と対応するように、getenv関数は文字列引数（環境変数と推定）を受け取り、変数の現在値を返します。変数が環境内に提供された名称で存在しない場合は'nil'を返します。

```
tempfile = getenv( "TMP" ) +  "/33j4ks.$$$" ;
```

hostVersion

hostVersion()関数は、ホストアプリケーションのバージョン番号を浮動小数点数値として返します。この数値はバージョンのメジャーとマイナー番号が入れられており、例えば6.1などとなります。

hostBuild

hostBuild()関数はアプリケーションのサブバージョンビルド番号を整数値で返します。LigthWaveのビルド番号は線形順でありリセットされることはありませんので、アプリケーションの機能セットを確定するにはバージョン番号よりもさらに正確に処理できます。

licenseId

licenseId()関数はドングル識別番号を返します。

```
dongleNum = licenseId();
```

platform

LScript のプリプロセッサは、プラットフォームと呼ばれる判別可能な状態を保持しています。この値には INTEL、ALPHA、SGI、MACINTOSH、SUN があります。'=='や'!='といった等号式で判別することが可能です。

```
@if platform == ALPHA
  info( "Alpha!" );
@end
@if platform == INTEL
  info( "Intel!" );
@end
```

platform

platform()関数は上記で解説した判別可能状態と同じ値を返しますが、マシンや OS を確定するためにランタイムでも使用が可能です。

```
x = platform();
```

runningUnder

runningUnder 関数は LAYOUT、MODELER、または SCREAMERNET のいずれかを返します。

```
app = runningUnder();
```

第 2 章：インターフェイスコマンド

この章では、レイアウトとモデラーの LScript 双方で利用可能なインターフェイスコマンドについて解説します。ここで紹介するのは作成、編集それにインターフェイス上のコントロールから情報を取得するための関数です。コントロールにはテキストやリスト、ボタンやタブ等があります。

これらのコマンドやパラメータのいくつかは **LScript Interface Designer（LSID）**を使用して自動的に作成することも可能ですが、手動で編集しなくてはいけない場合に備えて、使用法を読んだり理解できることが重要です。

メッセージ

info

`info()`コマンドは画面上にメッセージを表示します。このメッセージボックスは複数のデータタイプをサポートしています。

プロトタイプ：

```
void info(text)
  text      文字列; 表示されるメッセージ
```

例：

```
info( "hello world!" );
```

warn

`warn()`コマンドは画面上に警告メッセージを表示するもので、複数のデータタイプをサポートしています。スクリプトはコマンドを呼び出した後でも処理を続行します。

プロトタイプ：

```
void warn(text)
  text      文字列; 表示されるメッセージ
```

例：

```
warn( "Invalid Selection" );
```

error

`error()`コマンドは画面上にエラーメッセージを表示するもので、複数のデータタイプをサポートしています。スクリプトは`error()`コマンドを呼び出した後で終了します。

プロトタイプ：

```
void error(text)
  text      文字列; 表示されるメッセージ
```

例：

```
error( "Internal Error:  " + errorCode);
```

setdesc（レイアウトのみ）

`setdesc()`コマンドはレイアウトのプラグインパネル内に現在のスクリプトの解説文を設定します。単一の文字列を受け付けます。

プロトタイプ：

```
void setdesc(...text...)
  text      文字列; コンマで区切られた文字列リスト
```

例：

```
textValue = "Text3 ";
setdesc("Text1 ", "Text2 ", textValue);
```

進行モニター（モデラーのみ）

moninit

`moninit()`コマンドは、モニター制御コマンドのエントリーポイントです。`moninit()`はスクリプト用のモニターウィンドウを初期化します。LScript ではこのコマンドを他のモニター関数を使用する前に呼び出さなくてはいけません。

モニターシステムは基本的に進行表示を行います。進行表示はユーザーにスクリプトの処理情報を表示し、コンピュータが何らかの理由によりロック状態にあるのではないことを示します。作業終了までどのくらいのステップ数が必要となるのかを指定することでモニターを初期化し、この値を使用してモニターシステムは進行バーを表示します。進行バーはユーザーに処理が完了するまでにあとどのくらいの時間がかかるのかを示します。またモニターダイアログボックスの中でメッセージを表示することも可能です。

ユーザーはいつでもダイアログボックス内にある **Cancel** ボタンをクリックすることが出来ます。**Cancel** の動作は`monstep()`関数により扱われます（次に解説）。

プロトタイプ：

```
void moninit(steps [,message])
  steps       整数; 完了までのステップ数
  message     文字列; 表示されるメッセージ
```

例：

```
moninit(100, "Processing…");
```

monstep

`moninit()`関数で作成された進行バーを進めるため、`monstep()`関数を使用して距離を特定します。値が指定されていない場合には、進行カウントはデフォルトの 1 が設定されます。この値を指定することで、より大きな値で進行バーを進めることが出来ます。

`monstep()`はブール値を返します。大抵の場合、この値は`false`になりますが、ユーザーがモニターウィンドウ上の **Cancel** ボタンをクリックすると`true`を返します。この場合、スクリプトは出来るだけきれいな形で終了しなくてはなりません。

プロトタイプ：

```
Boolean monstep([step])
  step        整数; オプションの進行総数
```

例：

```
editbegin();
...
```

```
if(monstep())
{
  lyrsetfg(fg);
  lyrsetbg(bg);
  editend(ABORT);
  return;
}
```

monend

monend()コマンドはモニターシステムを終了します。モニターが開いている場合には進行ウィンドウを閉じ、モデラーの他の部分のモニターシステムを開放します（一度に一つのモニターウィンドウだけがアクティブになれます)。

プロトタイプ：

```
void monend(void)
```

リクエスタ

reqbegin

`reqbegin()`はリクエスタと呼ばれるユーザーからの情報やオプション値を集める入力ダイアログボックスを起動します。また情報やメッセージ（スタティックテキストとして）を伝えることも可能です。

`reqbegin()`を呼び出すと、リクエスタモードに代わります。このモードがアクティブになると、他のリクエスタに関連する関数を使用出来るようになりますが、終了する際は`reqend()`を呼び出します。リクエスタモードの最中でも任意にスクリプトを終わらせることが出来ますが、`reqend()`を呼び出し忘れた場合、LScript は自動的に`reqend()`を呼び出します。`reqbegin()`はリクエスタダイアログのタイトルを表す引数を一つ受け取ります。

プロトタイプ：

```
void reqbegin(title)
  title     文字列; ダイアログボックスのタイトル
```

例：

```
reqbegin( "My Requester" );
```

reqend

`reqend()`はリクエスタモードを終了します。`reqend()`を使用してユーザーの入力処理が終了した後は、リクエスタダイアログを終了させます。

プロトタイプ：

```
void reqend(void)
```

reqopen

リクエスタパネルをノンモーダルモードで開くためには、`reqpost()`の代わりに`req open()`関数が使用できます。ノンモーダルモードでは、パネルを開いたままで`options()` UDF が終了した後でさえもユーザーとインタラクティブにやり取りが出来ます。ノンモーダルモードでパネルコントロールの変更処理を行うために、コントロールにはアクティブなリフレッシュコールバック関数が適用されていなければなりません。コントロールが修正されると同時に、LScript からリフレッシュコールバック関数が呼び出され、その後スクリプトの`process()` UDF がレイアウトから呼び出されて、画面上でのオブジェクトの属性をリフレッシュします。

プロトタイプ：

```
result reqopen();
```

例：

```
reqbegin();
...
reqopen();
```

reqisopen

`reqisopen()`はノンモーダルで開かれているリクエスタパネルが現在も開かれているのかどうかを確定するのに使用されます。パネルが現在開かれていれば、ブール値`true`が返ります。

プロトタイプ：

```
Boolean reqisopen(void)
```

例：

```
if(reqisopen())
  reqend();
else
{
  reqbegin( "requester name" );
  ...
```

reqpost

`reqpost`コマンドは、リクエスタシステムにリクエスタダイアログの構築や要求されたフィールドへの入力が終了したことを伝えたり、ダイアログの表示などを伝えます。

`reqpost()`はユーザーが指定した値で処理を続行したいのかを示すブール値（`true`もしくは`false`）を返します。`true`であればユーザーが**OK**ボタンを押したことになりますので、処理を開始します。`false`であれば**Cancel**ボタンを押したことになりますので、速やかにスクリプトを終了させなくてはなりません。

`reqpost()`と`reqopen()`関数は、遊休時間（アイドル時間）処理を可能にするために二つの引数を受けとります。第1引数には呼び出されるUDFの名称を指定します。UDFは引数を何も受け付けず、リクエスタパネル上での描画を可能にします。オプションの第2引数はアイドル時間UDFの呼び出し間に使用されるタイムアウト値（ミリ秒）です。この値を省略するとデフォルトのタイムアウト値は500ミリ秒（2分の1秒）となります。

ある特定の状態下では、システムが正確な時間の間隔でUDFを呼び出すことが出来ない場合があります。このため、指定するタイムアウト値は近似値に過ぎないと考えておいてください。言い換えれば、アイドル時間UDFが呼び出された時点で指定した時間量が経過するだけであると仮定しておいてください。

プロトタイプ：

```
boolean reqpost(udf [,milliseconds])
  udf           udf;呼び出されるユーザー定義関数
  milliseconds  整数; タイムアウト間隔値
```

例：

```
generic
{
  reqbegin("Testing Idle");
    c1 = ctlnumber("Number",1.0);
    reqpost("idleTest"); // 500ミリ秒間隔で呼び出される
  reqend();
}
color = <0,255 * .5,255 * .75>;
idleTest
{
  drawline(color,0,2,100,2);
  color.x += 5;
  color.y += 5;
  color.z += 5;
  if(color.x > 255) color.x = 0;
  if(color.y > 255) color.y = 0;
  if(color.z > 255) color.z = 0;
}
```

reqabort

`reqabort()`関数はコントロールまたはリクエスタコールバック関数内部から呼び出され、即座にリクエスタを終了させます。ユーザーが終了処理を行う必要ありません。

`reqabort()`が引数無しで呼び出されると、"**Cancel**"の結果が`reqpost()`関数へと返されます。ユーザーが**Cancel**ボタンをクリックした状況をシミュレートしています。`reqabort()`が`true`の値を受け取った場合には、`reqpost()`に対し"**OK**"の結果が渡されます。

プロトタイプ：

```
void reqpost(state)
  state       状態; ブール値, true/false
```

例：

```
generic
{
  reqbegin( "Reqabort() Test" );
  reqsize(200,180);
    c1 = ctlstate( "Goodbye" ,0,50," goodbye" );
```

```
    ctlposition(c1,35,10);
    if(reqpost())
      info( "You pressed OK" );
    else
      info( "You pressed Cancel" );
  reqend();
}
goodbye: val
{
  reqabort(true);
}
```

requpdate

requpdate()関数は、リクエスタパネル上にあるリストボックスや情報コントロールをコントロールコールバック関数内部で使用したときに、新しい値で更新します。リストボックスのカウントまたはバリューコールバック関数内部からは、requpdate()関数を呼び出すことは出来ません。たとえ呼び出しても何の影響も及ぼしません。

プロトタイプ：

```
void requpdate(void)
```

例：

```
count;
lb_items;
generic
{
  for(count = 1;count <= 5;count++)
  lb_items +=  "Item_"  + count;
  reqbegin( "Testing requpdate()" );
    c1 = ctllistbox_ ( "Items" , 300, 300, " lb_count" , " lb_value" );
    c2 = ctlbutton( "Increment" ,200," inc_button" );
  reqpost();
}
lb_count
{
  return(lb_items.count());
}
lb_value: index
{
  return(lb_items[index]);
}
inc_button
{
  x = count;
```

```
    y = 1;
    count += 5;
    while(x <= count)
    {
      lb_items[y] = "Item_" + x;
      ++x;
      ++y;
    }
    requpdate();
}
```

reqsize

リクエスタパネルの幅と高さを手動で設定する場合は、`reqsize()`関数を使用してください。

プロトタイプ：

```
void reqsize(width,height)
  width      整数; パネルの幅
  height     整数; パネルの高さ
```

例：

```
reqsize(100,100);
```

reqresize

`reqresize()`関数を使用すると、ユーザーはインタラクティブにリクエスタパネルのサイズを変更することが可能になります。スクリプト内部で`reqresize()`を使用することでリサイズコールバック関数を定義することが出来ます。この関数は引数を 4 個受け取ります。指定する最初の文字列の値は、スクリプト内部で UDF を認識するための文字列です。

定義されたコールバック UDF はリクエスタパネルの新しい幅と高さを示す二つの整数引数値を受け取り、何も返しません。また二組のオプション引数はパネルの最小幅と高さ、最大幅と高さを提供します。これらの値が省略されている場合、最小幅と高さは初期ウィンドウの幅と高さとなり、最大幅と高さは画面解像度に限定されます。

以下の Generic スクリプトでは、サイズに関わりなくパネル上のある特定の位置にコントロールをロックすることでリサイズするメカニズムを解説しています。リクエスタパネルのリサイズは`reqresize()`関数がコールバック UDF の識別に成功したときにのみ可能になります。`reqresize()`が使用されない場合、パネルは初期サイズでロックされます。

プロトタイプ：

```
void reqresize(resize_udf,[minWidth,minHeight],[maxWidth, maxHeight])
  resize_udf  文字列; パネルがリサイズされるときに呼び出される UDF
  minWidth    整数; パネルの最小幅
  minHeight   整数; パネルの最小高さ
```

例：

```
c1;
generic
{
  reqbegin( "Resize Me!" );
    c1 = ctlnumber( "Number" ,1.0);
    reqresize( "resize" ,,,640,480); // 最大値は 640x480 になります
    reqpost();
  reqend();
}
resize: w,h
{
  c1.position(w - c1.w - 5,h - c1.h - 35);
}
```

reqredraw

reqredraw()関数はスクリプト内部においてコールバックUDFを確立します。このコールバック関数は、パネル自身が更新された時にはいつでも起動されます。

プロトタイプ：

```
void reqredraw(UDF)
  UDF         文字列; コールバック関数
```

例：

```
reqredraw( "callback" )
```

getvalue

getvalue()は入力フィールドの値を取得します。getvalue()へはコントロール作成関数の一つから返されるコントロールハンドルを指定します。戻り値のタイプはコントロールのタイプに依存します。習慣的に、getvalue()の戻り値は入力コントロールの初期化で使用されているオリジナルの変数へと割り当てられます。

プロトタイプ：

```
result getvalue(handle)
  result:    可変; フィールド内に含まれるデータ値
  handle:    整数; コントロールハンドル
```

例：

```
reqbegin( "Testing List Box" );
  c1 = ctllistbox( "Items" , 300, 300, " lb_count" , " lb_name" , " lb_event" );
  c2 = ctlbutton( "Select" ,50," button_event" );
  return if !reqpost();
  sel = getvalue(c1);
reqend();
```

setvalue

`setvalue()`を使用すると、既存のリクエスタコントロールの値を修正することが出来、`reqend()`を呼び出すまで必要であれば何度でも、ユーザーに対し入力用のリクエスタダイアログボックスを構築し表示することが可能です。`setvalue()`は変更するコントロールのハンドルと新しい値用の変数もしくは定数値を受け取ります。例としてLightWaveに配布されているlightswa.lsスクリプトをご参照ください。

プロトタイプ：

```
void setvalue(handle, value)
  handle:    整数; コントロールハンドル
  value:     データタイプ; 入力フィールドに対する新しい値
```

例：

```
setvalue(c1,a);
```

コントロール

以下のコントロール関数は、作成するコントロールIDの**Object Agent**を返します。データメンバやメソッドの完全なリストは、この後の「第４章：Object Agent Reference」を参照してください。

ctlstring, ctlinteger, ctlnumber, ctlvector

これらの関数を使用すれば、データタイプに対する編集コントロール、すなわち文字列、整数、浮動小数点数やベクトル（3つ1組の浮動小数点数の値）を作成することが出来ます。`ctlstring()`にはコントロールの幅を決定するための第3引数（オプション）が用意されています。

`ctlvector()`はかなり柔軟性のあるフォーマットで引数を受け取ります。それぞれの編集フィールド初期化するために、単一のベクトルデータ、三つのフィールド全てに対し適用する単一の数値、または三つの個別の値を渡すことが可能です。

プロトタイプ：

```
<Control Object Agent> ctlstring(label, value, [width])
<Control Object Agent> ctlinteger(label, value)
<Control Object Agent> ctlnumber(label, value)
<Control Object Agent> ctlvector(label, value)
   label     文字列; コントロールのラベル
   value     (コントロールに依存); コントロールの初期値
   width     整数; コントロールの幅
```

例：

```
name = "BillyBob";
age = 18;
c1 = ctlstring("Your name?",name);
c2 = ctlinteger("Your age?",age);
```

ctldistance

`ctldistance()`関数は`ctlnumber()`と似ていますが、編集コントロールの戻り値はモデラーの現在のデフォルト単位に基づいて値を表示または編集します。

プロトタイプ：

```
controlId ctldistance(label, value)
   label     文字列; コントロールのラベル
   value     数値; 初期値(距離)
```

例:

```
length = .25;
// 選択している初期単位がキロメーターの場合
// この編集フィールドは初期値として
// (.25 * 1k) == 250 メートルとなります
c1 = ctldistance( "Length" ,length);
```

ctlchoice

`ctlchoice()`を使うと、プッシュボタン選択コントロールが作成できます。この列挙選択リストは、同じ位置内にあるグループ化されたプッシュボタンとしてリクエスタシステムにより表示されます。

`ctlchoice()`はオプション引数1つを含む4つの引数を受け取ります。第1引数は他のコントロール作成関数と同様、コントロールのタイトルとなります。第2引数はパネルが開かれたときに選択されている初期プッシュボタンを特定する整数値です。第3引数にはボタンラベル用の文字列を参照する配列を指定することも出来ますし、同じデータを置くために直接、関数呼び出し内部にある初期化ブロックを指定しても構いません。選択番号は1から始まります。

オプションである第4引数で、選択リストの方向を指定することが出来ます。選択リストは水平方向または垂直方向に対しグループ化出来ます。このパラメータでは、垂直方向に対しブール値`true`を、水平方向に対し`false`を指定します。省略されるとリストはデフォルトで水平方向にグループ化されます。

プロトタイプ：

```
<Control Object Agent> ctlchoice(label,value, choices,[orientation]
  label       文字列; コントロールのラベル
  value       整数; デフォルト選択のインデックス値
  choices     文字列または配列; 選択リスト
  orienation  ブール値; true = 垂直   false = 水平
```

例:

```
booleans[1] =  "Yes" ;
booleans[2] =  "No" ;
choice = 1; // 初期選択して"No"を設定
// 垂直選択リストを作成
c1 = ctlchoice( "Quadrant?" ,choice,@" 90" ," 180" ," 270" @,true);
```

ctltext

`ctltext()`は、リクエスタダイアログボックスにスタティックテキストを追加します。このスタティックテキストを使うと特別な指示や入力値に関する詳細をユーザーメッセージとして表示することが出来ます。

`ctltext()`への引数は全て文字列ですが、引数の数は可変です。他のコントロール作成関数と同様、第1引数の文字列にはコントロールのラベルを指定します。また、参照配列を受け取り、文字列を最初の（そして唯一の）引数、または二番目（そして最後）の引数へと抽出することも可能です。

他のコントロール関数と同様、`ctltext()`はスタティックテキストコントロールのコントロールハンドルを返します。ただしこのコントロールは変更されることがないため、ハンドルはほとんど価値がありません。ダイアログボックスの再表示前に`setvalue()`関数からテキスト変更する場合に使用されるぐらいです。

プロトタイプ：

```
<Control Object Agent> ctltext(label, text[..text...])
```

`label`	文字列; コントロールのラベル
`text`	文字列または配列; 表示されるテキスト

例：

```
c1Text[1] = "Text1 ";
c1Text[2] = "Text2 ";
c1Text[3] = "Text3 ";
c0 = ctltext("Text1 ", "Text2 ", "Text3 ");
c1 = ctltext("Text1 ", c1Text);
```

ctlcolor

`ctlcolor()`は、三つの個別の入力編集フィールドを持つコントロールという点では`ctlvector()`と似たような動作を行います。しかし`ctlcolor()`の場合、入力フィールドには整数値だけしか入力できません。

また`ctlcolor()`でもパラメータとそのタイプの組み合わせを受け取ることが出来ます。単一のベクトルタイプ、三つ全てに同じ値を入れる単一の整数値、もしくは三つの編集フィールドそれぞれを初期化するための三つの整数値を指定することが可能です。

プロトタイプ：

```
ctlcolor(label, colorVal)
```

`label`	文字列; コントロールのラベル
`colorVal`	整数またはベクトル; 色の値

例：

```
c1 = ctlcolor( "Color" , 100);
または
c1 = ctlcolor( "Color" , <100,100,100>);
```

ctlsurface, ctlfont

ctlsurface()と ctlfont()関数は、それぞれモデラーのサーフェイスとフォント属性を管理するためのコントロールを作成します。作成されたコントロールは、それぞれのコントロールタイプの特定の機能と関連する他のコントロール自身をも保持する場合もあります。

例のごとく、どちらの関数も第 1 引数にはコントロールのタイトルを指定します。ctlsurface()へは第 2 引数にサーフェイスコントロールを初期化するためのサーフェイス名称を、また ctlfont()関数へは整数値を指定してください。整数値は現在読み込まれているフォントのインデックス値を指しており、コントロールの初期値となります。

プロトタイプ：

```
<Control Object Agent> ctlsurface(label, surfName)
      <Control Object Agent> ctlfont(label, fontIndex)
  label      文字列; コントロールのラベル
  SurfName   文字列; サーフェイス名称
  fontIndex  整数; フォントのインデックス
```

例：

```
c1 = ctlsurface( "surface" , "new surface" );
c2 = ctlfont( "font" , 1);
```

ctlpopup

ctlpopup()コマンドは、指定された値でポップアップコントロールを作成します。ポップアップはボタンと似たようなコントロールで、マウスで選択すると値のリストをスクロール表示します。

プロトタイプ：

```
<Control Object Agent> ctlpopup(label, value, choices[])
  label      文字列; コントロールのラベル
  value      整数; デフォルト選択のインデックス
  choices[]  配列; アイテムのリスト
```

例:

```
choice[1] =  "choice1" ;
choice[2] =  "choice2" ;
```

```
choice[3] =  "choice3" ;
c1 = ctlpopup( "Popup", 1, choice);
または
c1 = ctlpopup( "Popup" , 1, @" choice1" ,  "choice2" ,choice3" @);
```

ctlpercent

`ctlpercent()`は、値をパーセンテージで表示するミニスライダ付きの浮動小数点数の入力フィールドを作成します。`getvalue()`で浮動小数点数の値を返します。

プロトタイプ：

```
<Control Object Agent> ctlpercent(label, value)
   label     文字列; コントロールのラベル
   value     数値; 初期値
```

例：

```
c1 = ctlpercent( "Percent" , 100);
```

ctlangle

`ctlangle()`は、数値を角度（度数）で表示するミニスライダ付きの浮動小数点数の入力フィールドを持つコントロールです。`getvalue()`で浮動小数点数の値を返します。

角度の値はラジアン値であることに注意してください。度数で作業したい場合にはLScriptの`rad()`や`deg()`関数を使用して単位変換を行ってください。

プロトタイプ：

```
<Control Object Agent> ctlangle(label, value)
   label     文字列; コントロールのラベル
   value     数値; 初期値
```

例：

```
c1 = ctlangle( "Angle" , 180);
```

ctlrgb

`ctlrgb()`関数は、三つの数値入力フィールドをもつコントロールを作成します。ユーザーがこのフィールドにカラーコンポーネントを入力すると、プレビューエリアにはこれらの値を組み合わせた色が表示されます。

`label`はコントロールのラベルを表す文字列値であり、`rgb`は'x'、'y'、'z'の位置に赤、緑、青のカラーコンポーネントを表すベクトル値です。それぞれの値は範囲0～255の整数値です。`rgb`パラメータには全てのチャンネルで使用される単一の整数値、または各カラーチャンネルに対応する三つの整数値を指定することが可能です。

プロトタイプ：

```
<Control Object Agent> ctlrgb(label, rgb)
  label     文字列; コントロールのラベル
  rgb       整数もしくはベクトル; RGB の値
```

例:

```
c1 = ctlrgb( "Color(rgb)" , 10);
c1 = ctlrgb( "Color(rgb)" , <10, 10, 10>);
```

ctlhsv

`ctlhsv()`関数は、三つの数値入力フィールドをもつコントロールを作成します。ユーザーがこのフィールドにカラーコンポーネントを入力すると、プレビューエリアにはこれらの値を組み合わせた色が表示されます。

`label`はコントロールのラベルを表す文字列値であり、`hsv`は'x'、'y'、'z'の位置に色相、彩度、明度のカラーコンポーネントを表すベクトル値です。それぞれの値は範囲 0 ～ 255 の整数値です。`hsv`パラメータには全てのチャンネルで使用される単一の整数値、または各カラーチャンネルに対応する三つの整数値を指定することが可能です。

プロトタイプ：

```
<Control Object Agent> ctlhsv(label, hsv)
  label     文字列; コントロールのラベル
  hsv       ベクトル; 色相、彩度と明度の値
```

例：

```
c1 = ctlhsv( "Color (hsv):  ", <10, 30, 20>);
```

ctlcheckbox

`ctlcheckbox()`関数は、チェック状態をユーザーがオン/オフすることが可能なコントロールを作成します。

`label`はコントロールのラベルを表す文字列値です。`state`はチェックボックスの状態を示すブール値です。`true`は初期状態でコントロールがチェックされている状態（オン）、`false`はチェックをはずした状態（オフ）にします。

プロトタイプ：

```
<Control Object Agent> ctlcheckbox(label, state)
  label     文字列; コントロールのラベル
  state     ブール値; コントロールの状態を示す値(true = オン、false = オフ)
```

例：

```
c1 = ctlcheckbox( "checkbox" , true);
```

ctlstate

ctlstate()はチェックボックスと同じブーリアンボタンです。ctlbutton()と同様、スクリプト内部で事前定義、ユーザー定義関数（action_udf）でトリガーします。UDFがブーリアンコントロールの現在の状態を表す値を取得する点において、ctlbutton()とは異なります。

false（0）はオフの状態を、true（1）はオンの状態を指します。<width>はピクセル値で表現される点に注意してください。以下のスクリプトではctlstate()のセットアップ方を紹介しています。

プロトタイプ：

```
<Control Object Agent> ctlstate(label, value, width, action_udf)
  label       文字列; コントロールのラベル
  value       ブール値; コントロールの初期値
  width       整数; コントロールの幅
  action_udf  文字列; コントロール状態が変化したときに呼び出されるUDF
```

例：

```
@version 2.1
@warnings
generic
{
  reqbegin( "State Control" );
    c1 = ctlstate( "Testing" ,true,100," stateCallback" );
  reqpost();
}
stateCallback: val
{
  info(val); // 0 - オフ, 1 - オン
}
```

ctlfilename

このctlfilename()関数は、テキスト入力フィールドにファイルリクエスタダイアログボックスを出すボタンがついたコントロールを作成します。ファイルリクエスタから返される選択されたファイルは、テキスト入力フィールドに置き換わります。

labelはコントロールのラベルを表す文字列値です。filenameはコントロールのファイル名称編集フィールドの初期値を表す文字列値です。dialogは**保存**ダイアログに対してはブール値0（false）を、**開く**ダイアログに対してはブール値1（true）を指定します。

プロトタイプ：

```
string ctlfilename(label, filename[,width, [dialog]])
  label       文字列; リクエスタのタイトル
  filename    文字列; リクエスタ内の初期ファイル値
  width       整数; コントロールの幅
  height      整数; コントロールの高さ
  Dialog      ブール値; 0 = [保存] ダイアログ, 1 = [開く] ダイアログ
```

例：

```
filename = ctlfilename( "Open Object" , "(none)" , 70, 20);
```

ctlbutton

ctlbutton()は、スクリプト内部に事前にユーザーが定義する関数によってトリガーする"何らかの動作を行う"ためのボタンです。幅はピクセル値で指定します。

プロトタイプ：

```
<Control Object Agent> ctlbutton(label, width, action_udf)
  label       文字列; コントロールのラベル
  width       整数; コントロールの幅
  action_udf  文字列; ボタンが押されたときに呼び出されるユーザー定義関数
```

例：

```
@version 2.1
@warnings
c1..2;
generic
{
  reqbegin( "Testing" );
    c1 = ctlbutton( "Increment" ,50," addcount" );
    c2 = ctlinteger( "Count" ,1);
  reqpost();
}
addcount
{
setvalue(c2,getvalue(c2) + 1);
}
```

ctllistbox

ctllistbox()コントロールは、テキストエントリのグループを表示する単一列を表示します。スクリプト内部にはリストボックスのアイテム総数を返すためののユーザー定義関数（count_udf）とリストボックス内の指定されたインデックスオフセット値の場所にある文字列を返す関数（name_udf）の二つを定義しなくてはなりません。オプションとして event_udf も定義可能です。これはリストボックス内部でイベントがトリガーされたとき（例えばアイテムが選択されたときなど）にいつでもコントロールを受け取れるようにする関数です。

count_udf は引数を受け取ることなくアイテムの総数を整数値で返します。name_udf は照会される整数インデックス値を受け取り、スロット内部に置かれている文字列値を一つ返します。オプションの event_udf は選択されたアイテムのインデックス値の配列を受け取り、何も返しません。オプションの select_udf は選択されているエレメントのインデックスを受け取り、エレメントが選択可能かどうかを示すブール値 true/false を返します。

プロトタイプ：

```
<Control Object Agent> ctllistbox(label, width, height, count_udf,
name_udf_ [,event_udf,[select_udf]])
```

label	文字列; コントロールのラベル
width	整数; コントロールの幅
height	整数; コントロールの高さ
count_udf	文字列; LScript で使用されるアイテム総数を決定するユーザー定義関数
name_udf	文字列; LScript で使用されるリストボックスアイテムを取得するためのユーザー定義関数
select_udf	文字列; リストボックスのアイテムが選択可能かどうかをトグルするためのユーザー定義関数

例：

```
@version 2.2
@warnings
c1;
lb_items;
generic
{
  for(x = 1;x <= 5;x++)
  lb_items += "Item_" + x;
  reqbegin("Testing List Box");
    c1 = ctllistbox("Items",300,300, "lb_count","lb_name","lb_event");
    c2 = ctlbutton("Select",50,"button_event");
    return if !reqpost();
    sel = getvalue(c1);
  reqend();
  if(sel == nil)
```

```
    info("No selections were made");
  else
    info("You have selected '",sel,"'!");
}
lb_count
{
  return(lb_items.size());
}
lb_name: index
{
  return(lb_items[index]);
}
lb_event: items
{
  // 'items' は整数インデックス値の配列 もしくは 'nil'
  if(items == nil)
    info("No items are selected");
  else
    info("You have selected '",items,"'!");
}
button_event
{
  a = @1,3,5@;
  setvalue(c1,a);
}
```

加えて、以下のスクリプトではボタンコントロールがどのようにしてリストボックスのコンテンツを管理し、対話していくのかを紹介していきます。

```
@version 2.1
@warnings
c1..3;
lb_items;
main
{
  for(x = 1;x <= 5;x++)
  lb_items += "Item_" + x;
  reqbegin( "Testing List Box" );
    c1 = ctllistbox( "Items" ,300,10," lb_count" ," lb_name" );
    c2 = ctlbutton( "Add" ,200," add_button" );
    c3 = ctlbutton( "Delete" ,200," del_button" );
  reqpost();
}
```

```
lb_count
{
  // エレメントが'nil'であっても全エレメントをカウントしてしまうので
  // size()は使用しないで下さい
  return(lb_items.count());
}
lb_name: index
{
  return(lb_items[index]);
}
add_button
{
  lb_items += "Item_" + (lb_items.size() + 1);
  setvalue(c1,lb_items.count());
}
del_button
{
  sel = getvalue(c1);
  lb_items[sel] = nil;
  lb_items.pack();
  lb_items.trunc();
  setvalue(c1,lb_items.count());
}
```

ctlslider

ctlslider()関数は、ある特定の範囲付きスクロールバースライダーと数値入力フィールドを作成します。

プロトタイプ：

```
<Control Object Agent> ctlslider(title, val, min, max)
  title    文字列; コントロールのラベル
  val      数値; コントロールの初期値
  min      数値; 最小値
  max      数値; 最大値
```

例：

```
c1 = ctlslider( "My value" , 10, 0, 20);
```

ctlminislider

ctlminislider()関数は、即座に反応するスクロールボタン付きの数値入力フィールドを作成します。

プロトタイプ：

```
<Control Object Agent> ctlminislider(title, val, min, max)
  title    文字列; コントロールのラベル
  val      数値; コントロールの初期値
  min      数値; 最小値
  max      数値; 最大値
```

例:

```
c1 = ctlminislider( "My value" , 10, 0, 20);
```

ctlsep

`ctlsep()`を使用すると、リクエスタパネルを水平方向に横切る三次元の仕切り線を描画することが出来ます。`column`には仕切り線の開始列（ピクセル）、`width`には仕切り線の列数（ピクセル）を指定します。パラメータが指定されていない場合には、仕切り線はパネルの幅全体にまたがります。`width`が-1の場合、パネルの一番右端まで仕切り線を描画します。

プロトタイプ：

```
<Control Object Agent> ctlsep([column,width])
  column   整数; 開始列
  width    整数; バーの幅
```

例：

```
ctlsep();
c1 = ctlsep();
c1 = ctlsep(10, 10)
```

ctlimage

`ctlimage()`関数を使うと、パネルを元にしたリクエスタパネル上の任意の場所にTarga形式の画像ファイルを埋め込むことが出来ます（タイプ2と10、トップダウンもしくはボトムアップのラスター形式）。`ctlimage()`は必須の引数が一つと、それにオプションパラメータを三つ受け取ります。必須となるパラメータはイメージファイルの名称です。オプションとなるパラメータは画像のXオフセット（水平方向）、Yオフセット（垂直方向）、色透明度マスクまたは簡単な色フィルターとなる単一整数値もしくはベクトルです（場合によっては表示スピードが著しく遅くなる場合があります）。

この三つのオプションパラメータを使用する場合は、画像の拡大縮小も可能になります。第5引数（透明度ベクトルの後）に画像の幅の拡大縮小値を、第6引数に画像の高さの拡大縮小値を指定します。その後にくる最後のパラメータでは、画像のアスペクト比を保存するようLScriptに指示するブール値のフラグが指定できます。デフォルトではアスペクト比は保存されません。

LScript は、画像の幅と高さに対して指定された拡大縮小値の種類によって数値を判断します。どちらかに対し整数値を指定した場合、この値は画像の絶対幅（ピクセル値）であると考えます。一方、浮動少数点数を指定すると、パーセンテージであると解釈し（1.0 が 100% に相当）元の画像の幅または高さに対してパーセンテージを適用し、最終的なピクセル値を取得します。このメカニズムは画像の拡大に対しては設計されていないので、処理後の幅や高さが元の画像の大きさを越えるものであれば、オリジナルの大きさになります。

Image Editor を通して画像をレイアウトへと読み込む場合、この `ctlimage()` 関数を使用して表示することが可能です。画像名称の前にドルサイン記号（`$`）を挿入すると、LScript はディスク上の画像ファイルの位置を探しにいくのではなく、読み込まれている画像ファイルを使用することで名称を決定します。ドルサイン記号の後に続く名称は、レイアウトが画像に対し表示している名称と正確に一致していなければなりません。ランタイム実行形式でコンパイルするときには、スクリプトはこの形式で参照されている画像は無視してしまいます。つまりコンパイル済みのスクリプトを実行する際には、参照される画像はレイアウトから利用可能でなければならないのです。

プロトタイプ：

```
<Control Object Agent> ctlimage(imageName, xOffset, yOffset, transMask)
  imageName     文字列; 画像名称
  xOffset       整数; X オフセット座標値(水平方向)
  yOffset       整数; Y オフセット座標値(垂直方向)
  transMask     単一整数値もしくはベクトル; 簡易カラーフィルター
```

例：

```
...
// オリジナルの画像の 40% の幅、30% の高さで
// (0,0)に画像を表示します
ctlimage( "$ph_019.tga" ,0,0,,.4,.3);
...
// 幅/高さとも 100 ピクセル、アスペクト比保存で
// (0,0)の位置に画像を表示します
// (アスペクト比を保持する場合　正確に
// 100 × 100 ピクセルにはならない可能性があります)
ctlimage( "$ph_019.tga" ,0,0,,100,100);
...
// 画像を(0,0)の位置に修正無しで表示します
ctlimage( "$ph_019.tga" ,0,0);
ctlimage( "$ph_019.tga" ,0,0);
```

ctltab

`ctltab()`関数を使用すると、タブを作成することが出来ます。この関数では各タブのタイトルを表す文字列群を受け取ります。`ctlimage()`と同様、タブはLScriptコントロールの位置リリースロジックをトリガーすることなく、`ctlposition()`で配置が可能です。現在、パネル上でタブコントロールは一つだけ許可されています。

プロトタイプ：

```
<control Object Agent> ctltab(...tabNames...)
  tabNames 文字列; タブの名称リストはコンマで区切られます。
```

例：

```
c3 = ctltab( "Page 1","Page 2","Page 3" );
```

ctlallitems, ctlmeshitems, ctlcameraitems, ctllightitems, ctlboneitems, ctlimageitems, ctlchannel（レイアウトのみ）

これらの関数は、現在シーン中で利用可能なレイアウトのある特定タイプのアイテムをドロップダウン形式のリストとして提供します。各々のコントロールからは選択されたLightWaveオブジェクトを表す**Object Agent**が返されますが、オブジェクトが何も選択されていない場合には'`nil`'が返されます。

プロトタイプ：

```
<Object Agent> ctlallitems(label)
<Object Agent> ctlmeshitems(label)
<Object Agent> ctlcameraitems(label)
<Object Agent> ctllightitems(label)
<Object Agent> ctlboneitems(label)
<Object Agent> ctlimageitems(label)
<Object Agent> ctlchannel(label)
  label 文字列; コントロールのラベル
```

例：

```
obj = ctlmeshitems( "Objects in Scene:" );
```

コントロール管理

ctlpage

`ctlpage()`コマンドを使って、タブページ上にコントロールを配置することが可能です。このコマンドでは定義されたタブに相当するページ番号、単数もしくは複数のコントロールIDを受け取ります。コントロールがタブページ上に配置されると、タブがアクティブの場合にのみタブページ上に現れます。コントロールがタブページに関連付けられていない場合には、常にリクエスタパネル上で可視状態にあります。

また`ctlpage()`関数はコンマで区切ることで**Control Object Agents**の配列も受け付けることが可能です。参照配列は関数に対し第2（そして最後の）引数とする必要があります。

プロトタイプ：

```
void ctlpage(pageNum, … controlId's…)
  pageNum       整数; 定義されたタブ数
  controlId's Control Object Agent;コンマで区切られたControl Object Agentのリスト
```

例：

```
ctlpage(1,c4,c5);
ctlpage(2,c6);
ctlpage(3,c7,c8,c9);
```

ctlgroup

コントロールが`ctlgroup()`関数を使用してグループ化されている場合には、従属しているコントロールは全てグループのリーダーに対する相対位置に配置されます。最初の引数がグループリーダーとなるコントロールIDであり、後に続く引数は全てそのグループに属すコントロールとなります。

グループ化はコントロール配置設定よりも前に行わなければなりません。例えば、以下の関数ではタブコントロールに対し仕切り線を割り当てています。

プロトタイプ：

```
void ctlgroup(controlId, … controlId's…)
  controlIdControl Object Agent; グループリーダーとなるコントロール
  controlId'sControl Object Agents; コンマで区切られた以下のコントロールリスト
```

例：

```
c0 = ctltab( "Tab1","Tab2","Tab3");
c1 = ctlsep();
ctlgroup(c0,c1); // <- c0 はグループリーダー
```

```
ctlposition(c1,0,24); // <- c0 からの相対位置
ctlposition(c0,0,10); // <- グループリーダー c1 が後に続きます
グループ化も入れ子が可能です。
if(!reqbegin( "Panels Test" ,true))
  return;
c0 = ctlcheckbox( "Testing Number 1" ,true);
c1 = ctlcheckbox( "Testing Number 2" ,false);
c2 = ctlcheckbox( "Testing Number 3" ,true);
c3 = ctlcheckbox( "Testing Number 4" ,false);
ctlgroup(c0,c1,c2);
ctlgroup(c2,c3);
ctlposition(c0,10,10);
ctlposition(c1,0,20); // <- c0 からの相対位置
ctlposition(c2,0,60); // <- c0 からの相対位置
ctlposition(c3,0,20); // <- c2 からの相対位置
```

ctlposition

ctlposition()コマンドは、リクエスタパネル上で指定したX, Y座標値にコントロールを配置します。この関数を呼び出すと、LScriptは全てのコントロールに対しコントロールのフォーマットを放棄します。一旦この関数を呼び出してしまったら、リクエスタ上の全コントロールに対しこの関数を呼び出して位置を指定しないと、全てデフォルト位置（0,0）に配置されてしまい他のコントロールの上にどんどん重ねられてしまいます。

ctlposition()関数は三つのオプション数値引数を受け取ります。これらの引数でLScriptからリサイズ可能なリクエスタ上のコントロールへのアクセスが可能になります。第3引数はコントロールの幅（ピクセル値）を、第4引数は高さ（ピクセル値）を、第5引数はコントロールに対するオフセット値を指定します。オフセット値はラベル付きコントロールに対するラベルフィールドの幅を制御し、ラベル無しコントロールに対しては最左端からコントロールを右側へとずらす距離を制御することが出来ます。

プロトタイプ：

```
void ctlposition(controlId, column, row, [width],[height],[offset])
  controlIdControl Object Agent; 移動させるコントロール
  column      整数; X (水平)座標位置
  row         整数; Y (垂直)座標位置
  width       整数; コントロールの幅
  height      整数; コントロールの高さ
```

例:

```
ctlposition(c1, 5, 5, 70, 20, 10);
```

ctlactive

ctlactive()関数を使用すると、他のコントロール状態に基づいて単数もしくは複数のリクエスタコントロールを有効/無効にすることが出来ます。このメカニズムはLScriptリクエスタ用に作成されたものの中ではかなり複雑なものです。ctlgroup()と同様のパラメータ構造を持ちますが、ctlactive()への各呼び出しはLScriptのコールバックUDFから行われなければなりません。

関数パラメータは第1引数として"状態"コントロールIDを受け取ります（ctlgroup()におけるグループリーダーコントロールと同じです）。第2引数はカレントスクリプトでのUDFを識別する文字列です。残りの引数は全てこの状態に影響を受けるコントロールのIDとなります。

アクティブコールバックは状態が変化すると同時に、状態コントロールの値を保持する単一の引数を受け取ります。コールバックUDFは管理する全てのコントロールのアクティブ状況を示すブール値trueまたはfalseを返さなくてはなりません。ブール値trueは管理するコントロールを有効に、falseは無効にします。スクリプトはコールバックUDFに渡されたデータを評価し、適切な戻り値を決定しなくてはなりません。

例えば、状態コントロールがチェックボックスの場合は、コールバックUDFに渡される値はチェックボックスコントロールに等しいtrueまたはfalseとなります。ですからチェックボックスがtrueの場合に管理コントロールをアクティブにしておきたい場合には、単に渡された値を返すだけで構いません。しかし、状態コントロールが選択コントロール（choice）の場合、#2が選択されているときにのみ管理コントロールをアクティブにしたいかもしれません。

状態コントロールに対しては10個までコールバック関数を割り当てることが可能ですから、コントロールに対してより特定の起動操作を処理することが出来ます。ただしctlgroup()とは異なり、状態コントロールはカスケード状態にすることは出来ないため、管理パラメータリスト内で他コントロールに対する状態IDを含めることは出来ません。

プロトタイプ：

```
void ctlactive(controlId, active_udf, controlId)
  controlIdControl Object Agent; コントロールの状態
  active_udf       文字列; コールバックUDF.
  controlIdcontrol  ID; 影響を与えるコントロール
```

例:

```
generic
{
  x = 15;
  reqbegin( "ctlactive test" );
    c0 = ctlcheckbox( "This controls the status of ctlstring" , true);
    c1 = ctlstring( "ctlstring" ,"   ", 25);
    ctlactive(c0,  "toggleOn" , c1);
```

```
    ctlsep();
    c2 = ctlinteger( "Disperal Range" ,x);
    c3 = ctlcheckbox( "Testing Number 1" ,true);
    c4 = ctlchoice( "Test" ,2,@" X" ," Y" ," Z" @);
    ctlactive(c2," over10" ,c3);
    ctlactive(c2," under10" ,c4);
    return if !reqpost();
  reqend();
}
toggleOn: value
{
  return(value);
}
over10: value
{
  return(value > 10); //  'c3' をオンまたはオフに
}
under10: value
{
  return(value <= 10); //  'c4' をオンまたはオフに
}
```

ctlvisible

`ctlvisible()`関数では、いくつかの状態に基づいてコントロールの可視/非可視状態を制御することが可能です。呼び出しパラメータと操作要求は`ctlactive()`と同一です。`ctl visible()`、また第3（かつ最終）引数としてコントロールIDの配列を受け取ることも出来ます。

パネル上では、同じ位置/空間を占めていながら状況に応じて可視状態になるコントロールを組んでいくことで、本格的な外観を作り出すことが出来ます。このようにすれば、状況が変化するたびにコントロール自身が現れるようになります（例 #2参照）。

注意
LSIDE インターフェイスデザイナーは暗黙のうちにグループ化を行い使用していますが、ctlgroup()コマンドを扱う必要はありません

プロトタイプ：

```
void ctlvisible(controlId, visible_udf, ...controlId' s...)
  controlIdcontrol  ID; 管理コントロールの ID
  Visible_udf      文字列; 呼び出される UDF
  controlId's      配列; 影響を受けるコントロール
```

例 1 :

```
c0 = ctlchoice( "Test" ,2,@" Checkbox" ," Integer" @);
c1 = ctlcheckbox( "Testing Number 1" ,true);
c2 = ctlinteger( "Disperal Range" ,x);
...
ctlvisible(c0," showCheckBox" ,c1);
ctlvisible(c0," showInteger" ,c2);
...
showCheckbox: value
{
  return(value == 1);
}
showInteger: value
{
  return(value == 2);
}
...
```

例 2 :

```
c0 = ctlchoice( "Test" ,2,@" X" ," Y" ," Z" @);
c1 = ctlinteger( "Disperal Range" ,x);
c2 = ctlcheckbox( "Testing Number 1" ,true);
c3 = ctlpopup( "How  'bout a pop-up?" ,2,@" Item 1" ," Item 2" ," Item 3" @);
ctlvisible(c0," vis1" ,c1);
ctlvisible(c0," vis2" ,c2);
ctlvisible(c0," vis3" ,c3);
ctlposition(c0,10,10);
// コントロールの一番上に可視状態にあるコントロールをおきます
ctlposition(c1,0,30);
ctlposition(c2,0,30);
ctlposition(c3,0,30);
...
vis1: value
{ return(value == 1); }
vis2: value
{ return(value == 2); }
vis3: value
{ return(value == 3); }
...
```

ctlalign

`ctlalign()`関数は、インクリメンタルオフセットを使用して水平/垂直の値に対しマスターコントロールで整列させることが出来ます。第1引数にはオフセット開始位置となる座標位置を持つリードコントロール、その次に以下連続するコントロールに適用するXとYの整数インクリメント値を指定します。残りの引数は全てマスターコントロールに対しオフセット値で相対的に整列させられるコントロールです。

プロトタイプ：

```
void ctlalign(controlId, xOffset, yOffset, ...controlIds...)
  controlIdコントロールID; リードコントロール
  XOffset    整数; X オフセット
  YOffset    整数; Y オフセット
  controlIds Control Object Agent; 整列する全コントロール
```

例：

```
c0 = ctlchoice( "Test" ,2,@" X" ," Y" ," Z" @);
c1 = ctlinteger( "Disperal Range" ,x);
c2 = ctlcheckbox( "Testing Number 1" ,true);
c3 = ctlpopup( "How  'bout a popup?" , 2,@" Item 1" ," Item2" ," Item 3" @);
// c1/c2/c3をc0の位置からX オフセットを0から20ずつ
// 離しながら整列させます
ctlalign(c0,0,20,c1,c2,c3);
ctlposition(c0,10,10);
```

オフセット値はまた負の値でも指定できます。

```
ctlalign(c0,-5,20,c1,c2,c3); // c0 の左へ5単位ずつコントロールを移動します
ctlposition(c0,10,10);
```

注意
LSIDE インターフェイスデザイナーでは、このような手動のフォーマットを全てより洗練された形式で扱っています。

ctlrefresh

リクエスタ関数`ctlrefresh()`は、指定したコントロールのアクションをトリガーとしてパネル上の他のコントロールの値を更新することが出来ます。関数はコントロールID、リフレッシュコールバック関数の名称を受け取ります。特定のコントロールがユーザーによって修正されたとき（例えば、マウスのクリック、キーボード入力など）、リフレッシュコールバックはコントロールの現在の値と共に呼び出されます。他のリクエスタコントロールで使用されるコールバックとは異なり、`ctlrefresh()`用のコールバック関数は値を返す必要はありません。その代わり`setvalue()`関数を通して値を置き換えます。

プロトタイプ：

```
void ctlrefresh(controlId, refresh_udf)
  controlIdControl Object Agent; トリガーとなるコントロール
  refresh_udf 文字列; 更新が検出されたときに呼び出されるコールバック UDF
```

例：

```
filenames;
counter = 2;
c1;
main
{
  ...
  filenames = matchfiles( "c:\\temp" ," *.*" );
  c0 = ctlchoice( "Test" ,counter,@" X" ," Y" ," Z" @);
  c1 = ctlstring( "Filenames" ,filenames[counter]);
  ctlrefresh(c0," refresh" );
  ...
}
refresh: value
{
  setvalue(c1,filenames[value]);
}
```

大域変数に注意してください。リフレッシュコールバックはコントロールの値を直接修正するため、コントロールIDはグローバルレベルにおいて可視状態になくてはなりません。上記の例ではリフレッシュ関数が値を直接修正できるように変数`c1`をグローバル変数にしてあります。

ctlmenu

このコントロールタイプは、コントロールを押した時にポップアップメニューを表示するボタンから構成されています。第1引数はボタンのタイトル、その次は表示時にメニューに現れるアイテムのリストとなります。`ctlpopup()`と同様、このリストは配列参照でも値の初期ブロックでも可能です。このため要求されるコールバック関数はメニューからの選択による処理を定義されます。オプションである第4引数では個別のメニューエントリーの有効/無効化に使用されるコールバック関数を定義します。

特別エントリー値はメニュー仕切り線として解釈されます。最低二つの等記号（=）から始まるエントリーは、ポップアップメニューの指定された箇所に追加される仕切り線となります。各コールバック関数から返されるメニュー値は、オリジナルアイテム内における線形エントリー値に直接対応しています。しかし仕切り線エントリは処理されることはありえません。

プロトタイプ：

```
<Control Object Agent> ctlmenu(title, itemList, selection_udf,[enable_udf])
  title         文字列; リストボックスのラベル上に配置される文字
  itemList      配列または文字列; リストボックス内に描画されるアイテムのリスト
  selection_udf 文字列; リストボックス内のアイテムが選択されたときに呼び出される
                コールバック UDF の名称
  enable_udf    文字列; リストボックス内のアイテムが選択可能かどうかを確定するために呼び出される
                コールバック UDF の名称
```

例：

```
@version 2.4
@warnings
menu_items= @"New Session","Close Session","=====","Quit"@;
generic
{
  reqbegin("Testing");
    c1 = ctlmenu("Sessions",menu_items,"menu_select","menu_active");
    reqpost();
  reqend();
}
menu_select: item
{
  info("You selected'",menu_items[item],"'");
}
menu_active: item
{
  return(item != 2);
}
```

ctlinfo

`ctlinfo()`関数は、特定の UDF コールバック関数が呼び出された時点でリクエスタパネル上の領域に描画が発生するよう定義します。この領域で処理される描画は、全て LScript で切り取られます。`ctlinfo()`関数からは引数が三つ要求されます。第 1、第 2 引数には表示領域の幅と高さを指定します。第 3 引数は描画領域に再描画が必要となった場合に呼び出される UDF を定義します。この UDF は引数を受け取ることはなく、全てのリクエスタ再描画関数にアクセスします（`drawtext()`や`drawpixel()`など）。

プロトタイプ：

```
<Control Object Agent> ctlinfo(width, height, draw_udf)
  width      整数; 描画領域の幅
  height     整数; 描画領域の高さ
  draw_udf   文字列; 描画コールバック UDF の名称
```

例：

```
@version 2.4
@warnings
@define MSG "This is cool!"
msg_x = 101;
generic
{
  reqbegin("Testing");
    c1 = ctlinfo(100,30,"info_redraw");
    reqpost("marquee",50);
  reqend();
}
info_redraw
{
  drawbox(<132,130,132>,0,0,100,30);
  if(msg_x > 100)
    msg_x = -1 * drawtextwidth(MSG);
  drawtext(MSG,<0,0,0>,msg_x, integer((30 - drawtextheight(MSG))/ 2));
  drawborder(0,0,100,30,true);
}
marquee
{
  msg_x += 2;
  requpdate();
}
```

描画関数

パネル内でユーザー描画機能をサポートするリクエスタ関数がいくつか追加されました。描画関数はスクリプトの再描画コールバック関数内部でのみ呼び出し可能です。

reqredraw

`reqredraw()`関数は、パネルシステムからパネル自身が再描画される度に起動するスクリプト内部のコールバック関数を確立します。

プロトタイプ：

```
void reqredraw(redraw_udf)
  redraw_udf  文字列; UDF の名称
```

例：

```
reqredraw( "req_redraw" );
```

drawpixel

`drawpixel()`関数は、指定した色を使用してパネル上のある位置にピクセルを描画します。

プロトタイプ：

```
void drawpixel(<color>,xPos,yPos)
  <color>     ベクトル; ピクセルの描画色
  xPos        整数; ピクセルの X 座標値
  yPos        整数; ピクセルの Y 座標値
```

例：

```
drawpixel(<200,200,200>, 10, 10);
```

drawline

`drawline()`関数は、指定した色を使用してパネル上のある位置と位置の間に直線を描画します。

プロトタイプ：

```
void drawline(<color>, x1, y1, x2, y2)
  <color>     ベクトル; 直線の描画色
  x1          整数; 始点の x 座標値
  y1          整数; 始点の y 座標値
  x2          整数; 終点の x 座標値
  y2          整数; 終点の y 座標値
```

例：

```
drawline(<200,200,200>, 10, 10, 50, 50);
```

drawbox

drawbox()関数は、指定した色、幅、高さで塗りつぶし矩形を描画します。矩形は指定した位置に配置されます。

プロトタイプ：

```
void drawbox(<color>, xPos, yPos, width, height)
```

xPos	整数; 矩形始点の X 座標値
yPos	整数; 矩形始点の Y 座標値
width	整数; 矩形の幅
height	整数; 矩形の高さ

例：

```
drawbox(<200,200,200>, 10, 10, 50, 50);
```

drawborder

drawborder()関数は、指定した色、幅、高さで塗りつぶされていない三次元のボックスの輪郭を描画します。高さが0の場合には水平仕切り線が描画されます。drawborder()リクエスタ再描画関数は、描画されるボーダーの凹みなどを指定する5個のオプションブール値引数を受け取ります。この値が省略（もしくはfalse）されている場合はボーダーは“盛り上がって”見え、trueを指定するとボーダーは“へこんで”みえます。

プロトタイプ：

```
void drawborder(xPos, yPos, width, height, bool)
```

xPos	整数; 始点の X 座標値
yPos	整数; 始点の Y 座標値
width	整数; ボーダーの幅
height	整数; ボーダーの高さ
bool	ブール値; ボックスの3D外観の制御

例：

```
drawborder(5, 5, 100, 100, true);
```

drawtext

drawtext()関数は、指定した色で特定の箇所に提供された文字を描画します。

プロトタイプ：

```
void drawtext(text, <color>, xPos, yPos)
```

text	文字列; 描画する文字
<color>	ベクトル; 文字の描画色
xPos	整数; 文字を配置する x 座標値
yPos	整数; 文字を配置する y 座標値

例：

```
drawtext( "I'm drawing text", <200,200,200>, 10, 10);
```

drawerase

drawerase()関数は、リクエスタパネル上で消去する領域のＸとＹの位置、幅と高さを受け取ります。

プロトタイプ：

```
void drawerase(xPos, yPos, width, height)
  xPos        整数; 領域始点の x 座標位置
  yPos        整数; 領域始点の y 座標位置
  width       整数; 消去する領域の幅
  height      整数; 消去する領域の高さ
```

例：

```
drawerase(10,10,50,25);
```

第３章 ： モデラーコマンド

この章ではモデラースクリプトで利用可能なコマンド記述子を紹介します。ここには **MeshData Edit** と **CommandSequence** モードにあるコマンドがどちらも含まれています。この章で解説されているコマンドは、レイアウトでは動作しません。

コマンドリストの読み方

1. コマンド名称
2. 説明
3. プロトタイプ
   戻り値 コマンド（パラメータ[オプション]）（コマンドモード）

モデラーコマンドモード

CS … CommandSequence
MD … MeshDataEdit
IN … Internal

共通コマンド

new

`new()`コマンドは新規オブジェクトを作成します。

プロトタイプ：

```
new(Boolean) (CS)
```

undo, redo

`undo()`は、オブジェクトに対し処理された修正を元に戻すために呼び出されます。undo のレベル数は LightWave のモデラーコンフィグファイル（LWM3.CFG）で指定されたレベル数です。`redo()`は`undo()`関数が呼び出される前の状態にモデルを戻します。

プロトタイプ：

```
undo() (CS)
redo() (CS)
```

delete

`delete()`は選択されている前景レイヤーにあるポイント/ポリゴン情報を全て消去します。

プロトタイプ：

```
delete() (CS)
```

cut

`cut()`コマンドは、選択されているジオメトリを切り取り、モデラーの内部データクリップボードに保存します。他の多くのモデラーコマンドと同様、選択されている前景レイヤーでのみ動作します。

プロトタイプ：

```
cut() (CS)
```

copy

`copy()`コマンドは、選択されているジオメトリをコピーし、モデラーの内部データクリップボードに保存します。他の多くのモデラーコマンドと同様、選択されている前景レイヤーでのみ動作します。

プロトタイプ：

```
copy() (CS)
```

paste

`paste()`コマンドは、モデラーの内部データクリップボードからジオメトリを貼り付けます。

プロトタイプ：

```
paste() (CS)
```

load

`load()`コマンドでは、ディスク上にある LightWave のオブジェクトにアクセスすることが出来ます。このコマンドは選択されている前景レイヤーのデータに対し影響を及ぼします。

プロトタイプ：

```
status load(filename) (CS)
  filename    文字列; 読み込むファイル
```

save

`save()`コマンドでは、ディスク上にある LightWave のオブジェクトにアクセスすることが出来ます。このコマンドは選択されている前景レイヤーのデータに対し影響を及ぼします。

プロトタイプ：

```
status save(filename)(CS)
  filename    文字列; 保存するファイル
```

boundingbox

`boundingbox()`は、指定したレイヤー内にあるオブジェクトのバウンディングボックスを表す上限ベクトルと下限ベクトルを返します。レイヤーが特定されていない場合には、デフォルトとして選択されている前景レイヤーが使用されます。

`boundingbox()`は、モードに関わらずいつでも呼び出すことが可能です。しかし MeshData Edit の最中に使用されている場合、`boundingbox()`はレイヤーは指定することが出来ず、指定した値は無視されます。指定したレイヤーがポイント / ポリゴン情報を全く含んでいない場合には、'`nil`'が返されます。

プロトタイプ：

```
result boundingbox([layers]) (IN)
  戻り値      vector[2]; バウンディングボックスの上限/下限ベクトル
              データが存在しない場合には nil[2]
  layers      整数または整数配列; レイヤー番号の整数配列
              もしくはコンマで区切られたレイヤー番号
```

オブジェクト変形コマンド

fixedflex

`fixedflex()`関数を使うと、屈曲変形を伴うモデラーコマンドに対し操作パラメータを設定することが可能です。屈曲コマンドには`twist()`や`taper()`などがあります。`fixedflex()`は、これらの屈曲関数が特定の軸に沿って固定した範囲に対してのみ処理出来るように、操作パラメータを設定します。"i"や"o"フラグを使用して緩和状態（イーズイン/イーズアウト）を指定することが出来ます。

プロトタイプ：

```
status fixedflex(axis, start, end, easement) (CS)
  axis     定数; X, Y もしくは Z
  start    数値; 開始範囲(度数)
  end      数値; 終了範囲(度数)
  easement 文字列; "i" または "o".
```

autoflex

`fixedflex()`と同様、`autoflex()`関数は屈曲変形関数に対する操作パラメータを修正します。しかしこの関数では、指定した`polarity`の軸に沿って自動の範囲において屈曲コマンドを操作するようにパラメータを設定します。

プロトタイプ：

```
status autoflex(axis, polarity, easement) (CS)
  axis     定数; X, Y もしくは Z
  polarity 文字列; "+" または "- "
  easement 文字列; "i" または "o"
```

deformregion

`deformregion()`コマンドは、`vortex()`や`pole()`などの変形関数が使用する3次元空間の領域を確立します。オプションの軸が省略されている場合、変形効果は指定された半径から全方向に対し境界が区切られます。軸が指定されている場合、効果は特定の軸に対し境界がありません。

プロトタイプ：

```
status deformregion(radius [,center[,axis]]) (CS)
  radius   ベクトルまたは数値; 範囲の半径
  center   ベクトルまたは数値; 範囲の中心位置
  axis     定数; X, Y もしくは Z
```

move

`move()`は選択されているポイントを指定したオフセット分だけ移動させます。オフセット値は選択されているポイント群の中心点からの相対位置であり、原点からの距離ではありません。

プロトタイプ：

```
move(offset) (CS)
  offset    ベクトルまたは数値; 全ての軸に沿った移動オフセット値
```

プロトタイプ：

```
move(xoffset, yoffset, zoffset) (CS)
  xoffset     数値; X 軸に対するオフセット値
  yoffset     数値; Y 軸に対するオフセット値
  zoffset     数値; Z 軸に対するオフセット値
```

shear

`move()`と同じように、`shear()`は選択しているポイントを指定したオフセット値で変換します。しかしポイントは屈曲軸（`fixedflex()`もしくは`autoflex()`で設定された）に沿ってのみ変換されます。オフセット値は選択されているポイント群の中心点からの相対位置であり、原点からの距離ではありませんので注意してください。

プロトタイプ：

```
shear(offset) (CS)
  offset      ベクトルもしくは数値; 全ての軸に適用するオフセット値
```

プロトタイプ：

```
shear(xoffset, yoffset, zoffset) (CS)
  xoffset     数値; X 軸に対するオフセット値
  yoffset     数値; Y 軸に対するオフセット値
  zoffset     数値; Z 軸に対するオフセット値
```

magnet

`magnet()`もまた、選択しているポイントを指定したオフセット値で変換しますが、変形領域（`deformregion()`コマンドにより）で設定された領域内部で変形の度合いが減少します。

プロトタイプ：

```
magnet(offset) (CS)
  offset      ベクトルまたは数値; 全ての軸に適用するオフセット値
```

プロトタイプ：

```
magnet(xoffset, yoffset, zoffset) (CS)
  xoffset     数値; X 軸に対するオフセット値
  yoffset     数値; Y 軸に対するオフセット値
  zoffset     数値; z 軸に対するオフセット値
```

rotate

`rotate()`は選択しているポイントを指定した軸に沿って、指定した度数分だけ回転させます。

プロトタイプ：

```
rotate(angle, axis [,center]) (CS)
  angle     数値; 回転度数
  axis      定数; X, Y もしくは Z
  center    ベクトルまたは数値; 回転の中心位置
```

twist

`rotate()`と同様、`twist()`もまた、選択しているポイントを指定した軸に沿って、指定した度数分だけ回転させます。しかしポイントは屈曲軸（`fixedflex()`もしくは`autoflex()`コマンドで設定された）に沿ってのみ変換されます。

プロトタイプ：

```
status twist(angle, axis [,center]) (CS)
  angle     数値; ひねりの度数
  axis      定数; X, Y もしくは Z
  center    ベクトルまたは数値; ひねりの中心位置
```

vortex

`rotate()`と同様、`voxtex()`もまた、選択しているポイントを指定した軸に沿って、指定した度数分だけ回転させます。しかし変形領域（`deformregion()`コマンドにより）で設定された領域内部で処理が行われます。

プロトタイプ：

```
vortex(angle, axis [,center]) (CS)
  angle     数値; 渦巻きの度数
  axis      定数; X, Y もしくは Z
  center    ベクトルまたは数値; 渦巻きの中心位置
```

scale

`scale()`は、オプションとして指定する位置を中心として、指定した因数分だけ、選択しているポイントを拡大縮小します。任意の軸上で因数1であれば、その軸に沿った位置は保たれます。

例えばXとY軸に沿ってオブジェクトを二分の一に縮小し、Z軸に対しては現在の大きさを保持していたいのであれば以下のように指定します。

```
scale(<.5,.5,1>);
```

この例では、オプションとして中心位置が指定されていない点に注意してください。中心点はデフォルトで原点（<0,0,0>）になっており、値は全てこのポイントからの相対値となります。

プロトタイプ：

```
status scale(amount[,center]) (CS)
  amount    ベクトルまたは数値; 拡大縮小の因数
  center    ベクトルまたは数値; 拡大縮小の相対中心位置
```

taper

`taper()`は`scale()`と同様のコマンドですが、`fixedflex()`または`autoflex()`で設定されている屈曲軸を使用して影響範囲を定義している点だけが異なります。

プロトタイプ：

```
status taper(amount[,center]) (CS)
  amount    ベクトルまたは数値; 先細り因子
  center    ベクトルまたは数値; 先細りの相対中心位置
```

pole

`taper()`のように、`pole()`も`scale()`と同様のコマンドではありますが、変形領域（`deform region()`コマンドにより）で設定された領域内部に影響範囲を固定して変形を行います。

プロトタイプ：

```
status pole(amount [,center]) (CS)
  amount    ベクトルまたは数値; 磁極因子
  center    ベクトルまたは数値; 磁極の相対中心位置
```

bend

`bend()`コマンドは、オプションで設定する位置を中心点として、選択しているポイントを指定した角度だけ曲げるのに使用します。`bend()`は`fixedflex()`または`autoflex()`で設定された屈曲軸に固定されています。

プロトタイプ：

```
status bend(angle, direction [,center]) (CS)
  angle       数値; 屈折の度数
  direction   数値; 屈折方向(度数)
  center      ベクトルまたは数値; 屈折の相対中心位置
```

jitter

`jitter()`は、選択しているポイントを、4つの変位変換のうちの一つを使用してランダムに変換します。

`UNIFORM`は均一な範囲にポイントを振り分けます。これがデフォルトです。

`GAUSSIAN`は始点ポイントの回りの鐘形曲線状にオフセットを振り分けます。

`NORMAL`はローカルなサーフェイス法線に沿って内外にポイントを移動します。

`RADIAL`は`center`パラメータで指定した中心点を通る直線に沿って、内外にポイントを移動します。

プロトタイプ：

```
status jitter(radius [,type [,cntr]]) (CS)
  radius    数値; ジッターの半径
  type      定数; UNIFORM, GAUSSIAN, NORMAL, もしくは RADIAL
  cntr      ベクトルまたは数値; RADIAL で使用される中心位置
```

smooth

`smooth()`は、ポイントが連結していて影響を受けてしまうポリゴン内の“よじれ”を除去するために使用されます。

プロトタイプ：

```
status smooth([iterations [,strength]]) (CS)
  iterations  数値; 使用される反復回数
  strength    数値; スムースの強さ
```

quantize

`quantize()`コマンドを使用すれば、サイズパラメータで定義された三次元グリッドへ、選択されたポイントを全てスナップさせることができます。

プロトタイプ：

```
status quantize(size) (CS)
  size    ベクトルまたは数値; 三次元グリッドのサイズ
```

mergepoints

`mergepoints()`はポイント間の距離が最小距離内にある選択ポイントを統合するために使用されます。オプションである距離値が省略されている場合には、自動的に計算されます。

プロトタイプ：

```
status mergepoints([dist]) (CS)
  dist    数値; ポイント間の最小距離
```

オブジェクトコマンド

makebox

`makebox()`コマンドは、指定した範囲と分割数で三次元形状のボックスオブジェクトを作成します。

プロトタイプ：

```
status makebox(lowcorner, highcorner[,nsegments]) (CS)
  lowcorner   ベクトルまたは数値; 下限範囲ポイント
  highcorner  ベクトルまたは数値; 上限範囲ポイント
  nsegments   ベクトルまたは数値; ボックスのセグメント数
```

makeball

`makeball()`コマンドは三次元の天球状の球を作成します。

プロトタイプ：

```
status makeball(radius, nsides, nsegments[,center]) (CS)
  radius      ベクトルまたは数値; 球の半径
  nsides      整数; 球のサイド数
  nsegments   整数; 球のセグメント数
  center      ベクトルまたは数値; 球の中心位置
```

maketesball

`maketesball()`はモザイク状の球体を作成するのに使用します。モザイク状の球体は天球状の球と似ていますが、四面ポリゴンが三角ポリゴン化されています。`level`パラメータは適用するモザイクの量を確定します。

プロトタイプ：

```
status maketesball(radius, level[,center]) (CS)
  radius      ベクトルまたは数値; 球の半径
  level       整数; モザイクのレベル
  center      ベクトルまたは数値; 球の中心位置
```

makedisc

makedisc()コマンドを使用すると、指定したサイド、分割数を持つ円柱状のオブジェクトを作成できます。サイドの数が多ければ多いほど、円柱はより滑らかになります。

プロトタイプ：

```
status makedisc(radius, top, bottom, axis, nsides[,nsegments [,center]])
(CS)
```

radius	ベクトルまたは数値; radius of disc.
top	数値; 軸上における円柱の頂上
bottom	数値; 軸上における円柱の底面
axis	定数; X, Y もしくは Z
nsides	整数; サイド数
nsegments	整数; セグメント数 (1).
center	ベクトルまたは数値; オブジェクトの中心位置

makecone

makecone()コマンドを使用すると、指定したサイド、分割数を持つ円錐状のオブジェクトを作成できます。サイドの数が多ければ多いほど、円錐はより滑らかになります。makecone()は円柱の天頂を単一ポイントにして細くしたような形状です。

プロトタイプ：

```
status makecone(radius, top, bottom, axis, nsides[,nsegments [,center]])
(CS)
```

radius	ベクトルまたは数値; ディスクの半径
top	数値; 軸に沿った円錐の頂上
bottom	数値; 軸に沿った円錐の底面
axis	定数; X, Y もしくは Z
nsides	整数; サイド数
nsegments	整数; セグメント数 (1).
center	ベクトルまたは数値; オブジェクトの中心位置

maketext

maketext()関数は、文字列内の文字から構成されるモデラーオブジェクトを作成します。生成されるオブジェクトは、fontload()関数から読み込まれる文字の比率に基づきます。fontload()から返されるフォントインデックス値が、maketext()関数へと渡されることになります。

プロトタイプ：

```
number maketext(text, index, [,cornertype[,spacing[,scale[,axis[,pos]]]]])
(CS)
```

text	文字列; 変換する文字
index	整数; fontload()からのフォントインデックス
cornertype	定数; SHARP もしくは BUFFERED
spacing	数値; 文字間の間隔
scale	数値; 文字の拡大縮小因子
axis	定数; X, Y もしくは Z
pos	ベクトルまたは数値; 最初の文字の位置

以下のLScriptコードの断片は、これらの関数の使用法を紹介しています。

```
findex = fontload(fontName);
if(findex != nil)
{
   maketext( "LScript" ,findex);
```

maketext()は作成されるモデラーオブジェクトの幅（メートル単位）を返します。

複製コマンド

lathe

`lathe()`は、指定した軸に沿ってカレントのオブジェクトを回転させながら、指定したサイド数で新規オブジェクトを作成します。このオブジェクトには二次元のテンプレートオブジェクトを指定するのが一般的です。

プロトタイプ：

```
status lathe(axis, nsides[,center [, end [, start]]]) (CS)
  axis        定数; X, Y もしくは Z
  nsides      整数; 新規オブジェクトのサイド数
  center      ベクトルまたは数値; 回転体の中心点
  end         数値; 開始角度
  start       数値; 終了角度
```

extrude

`extrude()`は、指定した軸に沿ってオブジェクトを押し出していきます。オプションとしてセグメントを追加することも可能です。

プロトタイプ：

```
status extrude(axis, extent[,nsegments]) (CS)
  axis        定数; X, Y もしくは Z
  extent      数値; 軸に沿って押し出す長さ
  nsegments   整数; 追加するセグメント数(デフォルトでは 1)
```

mirror

`mirror()`コマンドは、平面を通して選択しているデータを 180 度回転した状態でコピーします。

プロトタイプ：

```
status mirror(axis, offset) (CS)
  axis        定数; X, Y もしくは Z
  offset      数値; 軸に沿った平面のオフセット値
```

pathclone, pathextrude

`pathclone()`と`pathextrude()`コマンドを使うと、LightWave 3D レイアウトで作成し保存したモーションパスに沿って、オブジェクトを複製したり押し出していくことが出来ます。オプションの`start`、`end`、`step`パラメータでループカウンタ用の引数が指定可能です。これらのオプション値を省略すると、モーションパスファイルに埋め込まれた開始と終了フレームが始点終点として使用され、デフォルトのステップ値は1となります。オプション値を指定することで、モーションパスファイル内にあるこれらの値を無効に出来ます。

始点と終点の値を変化させることで、これらの値が表すモーションパスの一部だけに沿って複製したり延ばすことが出来るようになります。例えば330フレームの長さを持つモーションパスが読み込まれた場合に、以下のようなコマンドを発行してみます。

例：

```
pathextrude( "my.mot",1,200,300);
```

`pathextrude( "my.mot",1,200,300)`はモーションパスの200フレーム目から300フレーム目までオブジェクトを押し出します。1～199フレーム、301～330フレームのオブジェクトを押し出すパスの部分には何も作成されません。

`pathclone()`を使用したオブジェクトの複製は、間隔毎にオブジェクトの複製を作成します。`pathextrude()`を使用してのオブジェクトの押し出しを選択すると、一つの新しいオブジェクトが各間隔に新規セグメントを持つオブジェクトを作成します。

どちらのコマンドも同じパラメータ数と種類を受け取ります。

プロトタイプ：

```
status pathextrude(filename, [,step[,start [,end]]]) (CS)
```

`filename`	文字列; モーションファイルの名称
`step`	数値; 作成するオブジェクト/セグメントの総数
`start`	数値; 開始点
`end`	数値; 終了点

railextrude, railclone

railclone()とrailextrude()は、オブジェクトを複製したり押し出したりという点ではpathclone()やpathextrude()とよく似ています。異なる点は、Railコマンドではモーションパスファイルの代わりにモデラーカーブを使用するという点です。これらのコマンドで使用されるカーブ（Rail）は、カレントのアクティブな背景レイヤーになくてはなりません。

プロトタイプ：

```
status railextrude(segs [,div [,flags [,strength]]]) (CS)
  segs        数値; セグメント数 (0=自動)
  div         定数; KNOTS もしくは LENGTHS
  flags       文字列; "o" (方向に沿う) もしくは "s" (スケール変更)
  strength    数値
```

soliddrill

soliddrill()は大半がaxisdrill()と同じですが、このコマンドではアクティブな背景レイヤーにある三次元オブジェクトを使用し、二つのオブジェクト（前景と背景）が重なり合う範囲が必要となります。

プロトタイプ：

```
status soliddrill(operation[,surface]) (CS)
  operation   定数; CORE, TUNNEL, SLICE もしくは STENCIL.
  surface     文字列; サーフェイス名称(STENCIL用)
```

axisdrill

axisdrill()はアクティブな背景レイヤーにある二次元オブジェクトをテンプレートとして使用し、アクティブな前景レイヤー内に“掘る”処理を行います。axisdrill()には4種類の掘り方が提供されています。

COREは背景のテンプレートの形状で前景のポリゴンを切り取ります。

TUNNELはCOREとは逆の処理です。テンプレートの形状で前景のポリゴンを削除し、掘り込まれていないデータが残ります。

SLICEはテンプレートのエッジに沿って前景オブジェクトに切り込みを作ります。

STENCILはSLICEと同じ処理を行いますが、テンプレートの形状内部にあるポリゴン全てに対し、サーフェイス名称を割り当てることが出来ます。

STENCILで掘るように指定している場合には、サーフェイス名称を指定するオプション引数が使用されます。

プロトタイプ：

```
status axisdrill(operation, axis[,surface]) (CS)
  operation     定数; CORE, TUNNEL, SLICE もしくは STENCIL.
  axis          定数; X, Y もしくは Z (STENCIL 用)
  surface       文字列; サーフェイス名称
```

boolean

boolean()コマンドは、アクティブな前景レイヤーにあるオブジェクトとアクティブな背景レイヤーにあるオブジェクトを掛け合わせます。ブーリアン処理には4種類がサポートされています。

UNION はそれぞれのオブジェクトを統合し一つの新規オブジェクトを作成します。二つのオブジェクトが重なり合う内部のポリゴンやサーフェイスは除去されます。

SUBTRACT は前景オブジェクトから背景オブジェクトの重なり合う部分を除去します。

INTERSECT は二つのオブジェクトが共有する部分からオブジェクトを作成します。

ADD は背景オブジェクトに前景オブジェクトを追加します。積算されたポリゴンは切り口が作成され、共有のエッジを作成します。

プロトタイプ：

```
status boolean(operation) (CS)
  operation     定数; UNION, SUBTRACT, INTERSECT もしくは ADD
```

bevel

bevel()コマンドを使うと inset と shift の量を指定することでオブジェクトにベベル状のエッジを追加することが出来ます。

プロトタイプ：

```
status bevel(inset, shift) (CS)
  inset         数値; ポリゴンの平面に沿って変形量
  shift         数値; ポリゴン平面に対する垂直の変位量
```

shapebevel

`shapebevel()`もまた、オブジェクトに対しベベル上のエッジを追加することが出来ます。ただし文字列内に一連の`inset`/`shift`値を受け取ります。文字列内には`inset`/`shift`の組がスペースによって区切られています。反復、あるいは連続した`bevel()`を使用するのと同じ効果が得られますが、自己交差が起こりうる可能性があるため、常に正しい結果が返るとは限りません。

以下のコードの断片では、このコマンドの使用法を解説しています。

プロトタイプ：

```
status shapebevel(pattern) (CS)
  pattern       文字列; inset/shift の組み合わせ
```

例：

```
s = string(-wide," ",wide," ",-wide," ",deep / 2);
shapebevel(s);
```

smoothshift

`smoothshift()`コマンドは、選択しているポリゴンを指定したオフセット分だけずらします。`maxangle`よりも大きな角度では、サーフェイスが壊れる可能性があります。

プロトタイプ：

```
status smoothshift(offset [,max [,scale]]) (CS)
  offset        数値; シフトオフセット値
  max           数値; 最大スムース角度
  scale         浮動小数点数; 拡大縮小因子
```

ポイント・ポリゴンコマンド

flip

flip()は、選択しているポリゴンのサーフェイス法線を反転するために使用されます。また、モデラーカーブの方向を反転するのにも使用出来ます。

プロトタイプ：

```
status flip(void) (CS)
```

triple

triple()を使用すると、選択したポリゴン全てを三角ポリゴンに変換します。特に4面ポリゴンを含んでいる場合、オブジェクトを分割する前にtriple()しておきたくなるかもしれません。以下のコードでの断片ではその部分を紹介しています。

プロトタイプ：

```
status triple(void) (CS)
```

例：

```
points = polycount();
if(points[5] or points[6])
// ポイント数4以上のポリゴンが存在します
triple();
subdivide(METAFORM);
```

freezecurves

freezecurves()は選択したモデラーカーブを全てポリゴンへ直接変換します。

プロトタイプ：

```
status freezecurves(void) (CS)
```

alignpols

alignpols()コマンドは、同じ方向を向くように（可能な限り）選択されたポリゴン全てを“反転”します。

プロトタイプ：

```
status alignpols(void) (CS)
```

removepols

`removepols()`は選択しているポリゴン全てを、ポイントを残したままで除去します。

プロトタイプ：

```
status removepols(void) (CS)
```

unifypols

`unifypols()`は重複しているポリゴンを全て削除します。

プロトタイプ：

```
status unifypols() (CS)
```

skinpols

`skinpols()`は、二つ以上のポリゴンをつなげて領域を囲むような三角形のメッシュを作成します。ロフティングと呼ばれることもあります。オリジナルのポリゴンが同じ頂点数を持つ必要はありません。

プロトタイプ：

```
skinpols() (CS)
```

mergepols

`mergepols()`は、選択しているポリゴンを単一ポリゴンへと統合します。選択されている各ポリゴンは、少なくとも一つ以上のエッジを他ポリゴンと共有している必要があります。

プロトタイプ：

```
mergepols() (CS)
```

subdivide

subdivide()は全ての三角ポリゴンをさらに小さな4個の三角形へと分割します。またこのコマンドでは、4角形をさらに小さな4個の四角形へと変換します。subdivide()では三つの異なる形式をサポートしています。

FLAT はポリゴンを分割し、オリジナルと同じ平面状に、新しく分割されたポリゴンを作成します。

SMOOTH は、分割した細部がオリジナルの形状の曲率を保持し拡張するように、補外処理を行います。

METAFORM はスムース処理をかけます。細部が大きくなっている箇所ではスムース処理を行い、よりきめ細かな細部にします。

プロトタイプ：

```
status subdivide(mode [,max]) (CS)
  mode        定数; FLAT, SMOOTH もしくは METAFORM
  max         数値; 最大スムース角度(度数)
```

fracsubdivide

fracsubdivide()はsubdivide()と似てはいますが、各新規頂点に対し、配置変数としてフラクタル変位値を適用することが出来ます。

プロトタイプ：

```
status fracsubdivide(mode, fractal [,max]) (CS)
  mode        定数; FLAT, SMOOTH もしくは METAFORM
  fractal     数値; フラクタル変位値
  max         数値; 最大スムース角度(度数)
```

pointcount

pointcount()は、アクティブ前景レイヤー内で選択されているポイントの個数を返します。この値はeditbegin()関数から返される値と等しくなります。pointcount()はいつでも呼び出すことが出来ます。

プロトタイプ：

```
result pointcount() (IN)
  result      整数; 選択されているポイント総数
```

polycount

`polycount()`の戻り値は、`pointcount()`の戻り値よりもさらに複雑です。ポリゴンに含まれている頂点数により返される値はグループ化しています。`polycount()`は六つの整数エレメントを返します。

1. 選択ポリゴンの総数
2. 選択されたポリゴンのうち、ポイントを1個だけ含むポリゴン数
3. 選択されたポリゴンのうち、ポイントを2個だけ含むポリゴン数
4. 選択されたポリゴンのうち、ポイントを3個だけ含むポリゴン数
5. 選択されたポリゴンのうち、ポイントを4個だけ含むポリゴン数
6. 選択されたポリゴンのうち、ポイントを4個よりも多く含むポリゴン数

このデータは、オブジェクトの状態を調べるために使用出来ます。例えば、オブジェクトに`METAFORM`をかける前には4面ポリゴンを全て三角形へと変換しておく必要があります。以下のコードの断片であれば、これが処理できます。

```
totalpoints[6];
totalpoints = polycount();
if(totalpoints[5] || totalpoints[6])
  triple();

subdivide(METAFORM);
```

プロトタイプ：

```
result pointcount() (IN)
  result      整数; 選択されているポイント数
```

weldpoints

`weldpoints()`は、選択しているポイント群を単一ポイントへと統合します。統合されたポイントの座標位置は、コマンドをかける前に最後に選択されたポイントの座標値となります。

プロトタイプ：

```
weldpoints() (CS)
```

splitpols

`splitpols()`は、ポリゴンを二つ以上の小さなポリゴンへと分割します。選択しているポイントの間に新しいエッジが作成されます。

プロトタイプ：

```
splitpols() (CS)
```

morphpols

`morphpols()`は、同じ頂点数を持つ二つのポリゴンを正確に連結する三角形のメッシュを作成します。新規メッシュは二つのオリジナルポリゴンを連結するラインに沿って指定したセグメント数へと分割されます。

プロトタイプ：

```
morphpols(nsegments) (CS)
  nsegments     整数; セグメント数
```

smoothcurves

`smoothcurves()`コマンドを使用すると、連結ポイント部分で二つのカーブの結合部分を滑らかにします。

プロトタイプ：

```
smoothcurves() (CS)
```

toggleCCstart, toggleCCend

`toggleCCstart()`と`toggleCCend()`は、カーブの開始と終了ポイントの解釈法に影響します。`toggleCCstart`で“オン”にトグルすると、カーブ上の開始（もしくは最終）ポイントとポイントは制御ポイントになります。このポイントとそこに連結しているカーブセグメントはカーブの一部である最初の（あるいは最後の）セグメントの形状に影響を与えます。

`toggleCCend`はポイントを“オフ”にトグルします。

プロトタイプ：

```
toggleCCstart() (CS)
toggleCCend() (CS)
```

togglepatches

`togglepatches()`を“オン”にトグルすると、選択したジオメトリをサブパッチのコントロールケージへと変換します。

プロトタイプ：

```
togglepatches() (CS)
```

make4patch

make4patch()を使用すれば、三つもしくは四つのカーブから四角形のメッシュを作成できます。垂直値、また水平値は、最後に選択した曲線に対し垂直または平行方向に作成されるセグメント数を設定します。カーブは頂点を共有し、閉じた領域を作らなくてはなりません。

プロトタイプ：

```
make4patch(nperpendicular, nparallel) (CS)
  nperpendicular  垂直セグメント数
  nparallel       水平セグメント数
```

close

close()を使用すると、アクティブオブジェクトを閉じます。アクティブオブジェクトが修正されている場合には、オブジェクトを除去してしまう前にメッシュデータを保存するかどうかをユーザーに問います。

プロトタイプ：

```
close() (CS)
```

closeall

closeall()を使用すると、モデラー内にあるオブジェクト全てを閉じます。オブジェクトが修正されていれば、除去する前に各メッシュを保存するかをユーザーに問います。

プロトタイプ：

```
cloesall() (CS)
```

exit

exit()はモデラーを終了します。オブジェクトが修正されていれば、除去する前に各メッシュを保存するかをユーザーに問います。

プロトタイプ：

```
exit() (CS)
```

swaphidden

swaphidden()は、カレントオブジェクトで隠されているメッシュデータを可視状態にし、可視状態にあるメッシュを隠します。

プロトタイプ：

```
swaphidden() (CS)
```

unweld

`unweld()`は、二つもしくはそれ以上のポリゴンがポイントを完全に共有しないように、頂点の共有を解除します。

プロトタイプ：

```
unweld() (CS)
```

weldaverage

`weldaverage()`は選択しているポイント群を三次元空間のその中心点へと統合します。

プロトタイプ：

```
weldaverage() (CS)
```

setobject

`setobject()`を使用すれば、アクティブなモデラーオブジェクトが選択できます。オプションとなるインデックス値は、同一名称をもつオブジェクトの中からどのオブジェクトを選択するのかを指定します。

プロトタイプ：

```
setobject(string, [ index ]) (CS)
  string    文字列; オブジェクト名称
  index     整数; 同じ名称をもつオブジェクトのうち選択するオブジェクトを特定するインデックス値
```

setpivot

`setpivot()`を使用すると、カレントオブジェクトのピボットポイントを特定の座標位置へと設定します。

プロトタイプ：

```
setpivot(vector) (CS)
  vector    ベクトル; 三次元空間内の座標位置
```

MeshDataEdit コマンド

editbegin

`editbegin()`は、LScript がサポートしている **MeshDataEdit** 関数へのエントリポイントです。**CommandSequence（CS/1.0）**コマンドを使用すると、ポイントやポリゴン群全体に対し処理を行うことが出来ますが、**MeshDataEdit（MD/1.0）**では、個々のポイントまたはポリゴンデータを直接編集することが出来ます。

`editbegin()`は LScript を **MeshDataEdit** モードへと切り替えます。このモードに入ると不思議な状態へと陥ります。

1. CommandSequence（CS/1.0）コマンドと関数は不当な処理となります。`editend()`を使用し MeshDataEdit を終了するまでは、CS タイプの処理は行えません。
2. 二つの新規自動 LScript 配列が利用可能になりました。一つ目の `points[]`配列には選択したオブジェクトポイント全てのポイント ID が、また二番目の `polygons[]`配列には、選択されているオブジェクトの全ポリゴンのポリゴン ID が含まれています。

自動配列 `points[]`と `polygons[]`の場合、**MeshDataEdit** モードのスコープ内部で同じ名称の配列や変数が宣言されていると、これらは隠されます。`editend()`を使用して **MeshData Edit** モードを終了した時点で、再び可視状態となります。

`editbegin()`は、現在選択されているポイントの総数を返します。これは自動配列 `points[]`のサイズと等しくなります。

プロトタイプ：

```
result editbegin() (MD)
  result:     整数; 選択されているポイント総数
```

editend

`editend()`を使用して、**MeshDataEdit** モード（`editbegin()`によって初期化された）を終了します。この関数が呼び出されると自動配列 `points[]`と `polygons[]`は消去され、**Command Sequence（CS/1.0）**関数とコマンドが再び利用可能になります。

MeshDataEdit 操作の間、モデラーはオブジェクトデータに施された修正全てを“待ち行列に入れて”あります。`editend()`コマンドを発行するまでは、修正は適用されないのです。デフォルトでは、`editend()`に何もパラメータを指定しない場合は、修正が適用され処理が続行します。しかし、待ち状態にある変更を破棄するためのパラメータを `editend()`に渡した場合には、オブジェクトは修正されないままとなります。このパラメータに定数 `ABORT`、または非ゼロを指定することで、全ての修正を破棄するかどうかを指定します。

プロトタイプ：

```
editend([status]) (MD)
  status    定数; ABORT.
```

addpoint

addpoint()はオブジェクトに対し新規ポイントデータを追加します。有効なベクトルを指定すると、addpoint()は他の関数へ渡すことが可能となる有効なポイントIDを返します。何らかの理由でaddpoint()が新規ポイントの作成に失敗すると、'nil'が返されます。

プロトタイプ：

```
result addpoint(location) (MD)
  result      ポイントID; 新規頂点のポイントID
  location    整数配列[3],ベクトルもしくは数値; 新規ベクトルの座標位置
```

addpolygon

addpolygon()を使用すると、オブジェクトデータに対して新規ポリゴンを追加できます。この関数に対し単数または複数のポイントIDを指定することで、新規ポリゴンを生成できます。複数のポイントIDを指定する場合には配列形式にしてください。

新規ポリゴンには、サーフェイス名称を割り当てることが出来ます。サーフェイスパラメータが省略されている場合、新規ポリゴンにはデフォルトのサーフェイスIDが割り当てられます。指定したサーフェイス名称が存在しない場合には、そのサーフェイス名称も作成されます。

この関数が作成した場合にはポリゴンIDが、失敗した場合には'nil'が返されます。

プロトタイプ：

```
result addpolygon(points [,surface]) (MD)
  result      ポリゴンIDまたはnil; 新規ポリゴンID
  points      ポイントID配列[]もしくはポイントID; 新規ポリゴン用のポイントID
  surface     文字列; ポリゴンに割り当てられるサーフェイス名称
```

addcurve

addcurve()はパラメータタイプ、パラメータ総数ともにaddpolygon()と同一です。しかし、addcurve()は指定されたポイントをポリゴンの代わりにモデラーカーブへと変換します。ポリゴンとカーブの違いは、ポリゴンにおいてポイント間の角度がきっちりと直線状になっているのに対し、カーブのポイントは滑らかに補完されている点にあります。閉じたカーブを作成するためには、始点と終点のポイントを重複させる必要があります。

プロトタイプ：

```
result addcurve(points [,surface [state]]) (MD)
  result    ポリゴン ID または nil; 新規カーブの ID
  points    ポイント ID 配列[]またはポイント ID; 新規カーブのポイント ID
  surface   文字列; カーブに割り当てるサーフェイス名称
  state     定数;カーブの制御構造を指定するフラグ START もしは END
            パラメータが指定されていない場合にはカーブに適用されるコントロールはありません
```

addquad

addquad()は、四角形（4 頂点）ポリゴンをオブジェクトのメッシュデータへと追加します。ポイント ID は個別に、または要素 4 の配列として指定出来ます。

プロトタイプ：

```
result addquad(points) (MD)
  result    ブール値; 0 == 成功
  points    ポイント ID 配列[4]もしくは個別のポイント ID; 4 面ポリゴン構成に使用されるポイント
```

addtriangle

addtriangle()は、パラメータタイプという点では addquad()と似通っていますが、新規ポリゴン構築に際し、4 点の代わりに 3 点を使用します。

プロトタイプ：

```
result addtriangle(points) (MD)
  result    ブール値; 0 == 成功
  points    ポイント ID 配列[3]もしくは個別のポイント ID; 3 面ポリゴン構成に使用されるポイント
```

polyinfo

polyinfo()関数を使用すると、個々のポリゴンに関する情報を集めることが出来ます。polyinfo()はアイテムの変数を返します。この変数の最初のエレメントは常にポリゴンに割り当てられているサーフェイス名称となり、残りの要素はポリゴンを構成しているそれぞれのポイント ID となります。

オブジェクト内にある全てのポリゴンが同一数のポイントを持っているわけではないので、ポリゴンのポイント数はポリゴンによって異なります。ポリゴンに含まれるポイント数を決定するためには、polypointcount()関数（次に解説）を使用することで、polyinfo()から返される値を保持するための適切なサイズを持つ配列を割り当てることが出来ます。

プロトタイプ：

```
result polyinfo(polygon) (MD)
  result      配列[]; 最初の要素はサーフェイス名称、
              残りの要素にはポイント ID が入っている可変サイズの配列
  polygon     ポリゴン ID; 情報を取得したい対象のポリゴン ID
```

polynormal

`polynormal()`は指定したポリゴンのサーフェイス法線を表すベクトルを返します。

プロトタイプ：

```
status polynormal(polygon) (MD)
  status      ブール値; 操作が成功した場合
  polygon     ポリゴン ID; 処理を施すポリゴン
  surface     文字列; 現在ポリゴンに割り当てられているサーフェイス名称
```

polypoints

`polypoints()`を使用すると、ポリゴンに新しいポイントセットを割り当てることが出来ます。ポリゴンを構成しているオリジナルのポイントは自動的には除去されませんので、必要でない場合には手動で除去する必要があります（`rempoint()`を使用）。

プロトタイプ：

```
status polypoints(polygon, points) (MD)
  status      ブール値; 処理が成功した場合
  polygon     ポリゴン ID; 処理を施すポリゴン
  points      ポイント ID 配列[]; ポリゴン用の新規ポイント
```

rempoint, rempoly

`rempoints()`と`rempoly()`はそれぞれ、ポイントやポリゴンのオブジェクトのメッシュデータから除去する関数です。`rempoly()`の場合には、ポリゴンを構成しているポイントが除去されることはなく、メッシュデータの一部としてそのまま残ります。

プロトタイプ：

```
status rempoint(point) (MD)
  point ID    除去するポイント
```

プロトタイプ：

```
status rempoly(polygon) (MD)
  polygon ID 除去するポリゴン
```

pointmove

`pointmove()`はポイントの位置ベクトルを変更します。

プロトタイプ：

```
status pointmove(point, location)
  point ID    移動するポイント
  location    整数配列[3], ベクトルもしくは数値; ポイントの新規座標位置
```

polysurface

`polysurface()`を使用すると、LScript はポリゴンのサーフェイス名称を再び割り当てることが出来ます。

プロトタイプ：

```
status polysurface(polygon,surface)
  polygon     文字列; 新たにポリゴンに割り当てるサーフェイス名称
  surface     ポリゴン ID; 修正を施すポリゴン
```

polypointcount

`polypointcount()`は、`polyinfo()`関数と共に使用することで、ポリゴンに含まれるポイント数を決定することが出来ます。一般的に`polyinfo()`から返される情報を保持するための配列サイズを決定するときに、この情報が必要となります。

```
...
pcount = polypointcount(polygons[x]);
ppoints[pcount + 1]; // サーフェイス名称[1]を考慮
ppoints = polyinfo(polygons[x]);
...
```

プロトタイプ：

```
result polypointcount(polygon) (MD)
  result      整数; ポイント数
  polygon     ポリゴン ID; 照会するポリゴン
```

pointinfo

`pointinfo()`はポイントが存在する三次元空間内の位置を返します。

プロトタイプ：

```
result pointinfo(point) (MD)
  result      ベクトル; ポイント座標位置
  point       ポイント ID; 照会するポイント ID
```

サーフェイスコマンド

setsurface

`setsurface()`は、全新規データが割り当てられるサーフェイス名称を設定します。存在しない場合には作成されます。

プロトタイプ：

```
status setsurface(name) (CS)
  name      文字列; 新規サーフェイス名称
```

getdefaultsurface

`getdefaultsurface()`コマンドは、選択されているサーフェイス名称を返します。

プロトタイプ：

```
string getdefaultsurface(void) (IN)
```

changesurface

`changesurface()`は、全選択ポリゴンに新規または既存のサーフェイス名称を割り当てます。

プロトタイプ：

```
status changesurface(name) (CS)
  string    文字列; 選択しているポリゴンに割り当てるサーフェイス名称
```

nextsurface

`nextsurface()`を使用すると、モデラーの内部サーフェイス名称リストを通して照会していくことが出来ます。パラメータ無しで`nextsurface()`を呼び出すと、モデラーリストの最初のサーフェイス名称が返されます。`nextsurfae()`に前のサーフェイス名称を引数として渡すことで、次のサーフェイス名称が返されます。リストの終端にあるサーフェイス名称が渡されると'`nil`'が返されます。

プロトタイプ：

```
result nextsurface([name]) (IN)
  result    文字列; サーフェイス名称
  name      文字列; オプションサーフェイス名称
```

例：

```
// カレントのサーフェイス名称全てを表示します
names
surface = nextsurface();
info(surface);
while(true)
{
  if((surface = nextsurface(surface)) == nil)
    break;
  info(surface);
}
```

renamesurface

`renamesurface()`を使用すると、既存サーフェイスの名称を変更できます。一度変更すると、オリジナルのサーフェイス名称は存在しなくなります。

プロトタイプ：

```
status renamesurface(orig, new) (IN)
  orig      文字列; 既存のサーフェイス名称
  new       文字列; 新規サーフェイス名称
```

createsurface

`createsurface()`コマンドは、モデラーの内部リストに新規サーフェイス名称を追加します。この名称を持つサーフェイスが既に存在していたら、何も起こりません。

プロトタイプ：

```
status createsurface(name) (IN)
  name      文字列; 新規サーフェイス名称
```

copysurface

`copysurface()`コマンドは、モデラーの内部リストに新規サーフェイス名を追加し、既存のサーフェイスの属性をそれにコピーします。

プロトタイプ：

```
status copysurface(orig, new) (IN)
  orig      文字列; 既存サーフェイス名称
  new       文字列; 新規サーフェイス名称
```

フォントコマンド

fontcount

`fontcount()`は、現在モデラーに読み込まれているフォント数を返します。内部フォントリストが空の場合には'`nil`'が返ります。

プロトタイプ：

```
result fontcount(void) (IN)
  result    整数; 読み込まれているフォント数
```

fontindex

フォント名称を指定すると、`fontindex()`はフォントのインデックス値を返します。指定したフォントが読み込めない場合には、'`nil`'を返します。

プロトタイプ：

```
result fontindex(name) (IN)
  result    整数; フォントインデックス
  name      文字列; フォント名称
```

fontname

`fontname()`はフォントインデックス（`fontload()`や`fontindex()`で返された値）が指定された場合、特定のフォント名称を返します。指定されたインデックス値が無効な場合には'`nil`'が返ります。

プロトタイプ：

```
result fontname(index) (IN)
  result    文字列もしくは nil; フォント名称
  index     整数; フォントインデックス値
```

fontload

`fontload()`はポストスクリプトフォントをモデラーに読み込み、テキスト生成コマンドで使用可能にします。同類のコマンドである`fontclear()`（次で解説）は、特定フォントが終了した段階で呼び出されます。

プロトタイプ：

```
result fontload(name) (IN)
  result    整数; フォントインデックスもしくは nil
  name      文字列; オプションパス付きのフォント名称
```

例：

```
fontName = getfile( "Select a Font" );
if(fontName == nil)
  return;
findex = fontload(fontName);
if(findex != nil)
{
  maketext( "LScript" ,findex);
  fontclear(findex); // 即座にフォントを開放
```

fontclear

`fontclear()`は、`fontload()`コマンドで読み込まれているモデラーフォントが必要なくなった段階で呼び出します。

プロトタイプ：

```
status fontclear(index) (IN)
  index     整数; フォントインデックス値
```

プラグイン

cmdseq

`cmdseq()`は、LScriptがモデラーにインストールされている他のプラグインを呼び出せるようにします。他プラグインへのインターフェイスは一方向性のものです。つまり、これらプラグインに対し情報を渡すことは出来ますが、戻り値が返されることはありません。

他のプラグインを呼び出すために使用される名称は**Custom**メニューボタン内に表示される名称と同一です。プラグインに渡される引数は、どのようにプラグインがプログラムされているかによります。特別にこの形式で呼び出されるためにデザインされたプラグインを除くと、パラメータを受け取らないプラグインが大半です。

プロトタイプ：

```
status cmdseq(name[,arguments]) (CS)
  name        文字列; プラグイン名称
  arguments   文字列; プラグインに依存する可変値
```

レイヤーコマンド

lyrfg, lyrbg

`lyrfg()`と`lyrbg()`コマンドはそれぞれ、選択されているモデラーの前景レイヤーと背景レイヤーを返します。一般的にこれらコマンドの各戻り値には配列を割り当てますが、連想割り当てを使用して非配列変数を割り当てることも可能です。

通常、LScriptが何らかの設定で修正し始める前に、このコマンドを使用してアクティブに選択されているレイヤーを記憶しておきます。処理を実行したら、`lyrsetfg()`（配列がアクティブな背景情報を含んでいる場合には`lyrsetbg()`）に値を渡すことで、ユーザーのオリジナルレイヤーに設定しなおします。

プロトタイプ：

```
result lyrfg() (IN)
result lyrbg() (IN)
  result    整数配列[]; 単数もしくは複数のレイヤー値
```

lyrdata, lyrempty

`lyrdata()`と`lyrempty()`関数は、メッシュデータを含むレイヤーの番号を表す配列（`lyrdata`）と全く含まないレイヤー番号の配列（`lyrempty`）を返します。

これらの関数は、レイヤーがアクティブな前景でも、アクティブな背景でも、全く選択されていなくてもレイヤー番号を返します。これらの関数のプロトタイプは以下のとおりです。

プロトタイプ：

```
result lyrdata() (IN)
result lyrempty() (IN)
  result    整数配列[]; 単数もしくは複数のレイヤー値
```

lyremptyfg, lyremptybg

`lyremptyfg()`と`lyremptybg()`はそれぞれ、選択されている前景と背景レイヤーのうち空レイヤーを返します。

プロトタイプ：

```
result lyremptyfg() (IN)
result lyremptybg() (IN)
  result    整数配列[]; 単数もしくは複数のレイヤー値
```

lyrswap

`lyrswap()`は、選択されているレイヤーを瞬時に反転します。選択されている前景レイヤーは背景レイヤーとなり、選択されている背景レイヤーは開くアクティブな前景レイヤーとなります。

このコマンドはモデラーにおいてレイヤー反転に使用する引用符（'）に相当する機能です。

プロトタイプ：

```
lyrswap() (IN)
```

lyrsetfg, lyrsetbg

`lyrsetfg()`と`lyrsetbg()`関数は、整数値の配列またはアクティブとして選択されているレイヤーを示す整数値の変数を受け取ります。

プロトタイプ：

```
lyrsetfg(lyr1[,lyr2...lyr20]) (IN)
lyrsetbg(layers) (IN)
  lyr1          整数; レイヤー番号
  lyr2..lyr20   追加レイヤー
  layers        整数配列[]; 要素はレイヤー値
```

setlayername

`setlayername()`は、選択されている前景レイヤーに名称を設定します。引数無しでこの関数を呼び出すと、それまで設定されていたレイヤー名称を除去します。

プロトタイプ：

```
setlayername([layerName]) (CS)
  layername     文字列; レイヤー名称用の文字列
```

選択コマンド

selpoint

他の全ての LScript コマンドの中で（次に解説する `selpolygon()` を除き）、この `selpoint()` は最も複雑なパラメータ構造をもつコマンドです。`selpoint()` は二つの “ メジャーモード ” と多数の “ マイナーモード ” をもちます。これらのモードはそれぞれに独自のパラメータ数とパラメータタイプを持ちます。

メジャーモード：

`SET`: この後にマイナーモードが続くと選択するポイントを定義します。

`CLEAR`: この後にマイナーモードが続くと選択解除するポイントを定義します。

上記メジャーモードのどちらか一つが、常に `selpoint()` の第 1 引数となります。これらはコマンドに対する唯一のオプション無しのパラメータであり、パラメータ自身でリストに表すことが可能です（例えば `selpoint(CLEAR)` は全ての選択されているポイントの選択を解除します）。

`selpoint()` でサポートしているマイナーモードを使うことにより、いくつもの方法でポイント選択を指定できます。`selpoint()` に対するマイナーモードを以下に示します。

マイナーモード：

`VOLUME`

引数： バウンディングボックスの上限/下限を表す二つのベクトル

解説： 指定したバウンディングボックス内部に存在するポイントの選択/解除を行います。

`CONNECT`

引数： なし

解説： 既に選択されているポイントに連結しているポイントを選択します。メジャーモード `SET` でのみ動作します。

`NPEQ`

引数： ポリゴンに含まれるポイント数

解説： 引数で指定されたポイント数分だけを含むポリゴンに属しているポイント全てを選択/解除します。

`NPLT`

引数： 限定するポリゴン内におけるポイント数

解説： 引数で指定されるポイント数よりも少ないポイント数で構成されるポリゴンに属しているポイント全てを選択/解除します。

NPGT

引数：限定するポリゴン内におけるポイント数

解説：引数で指定されるポイント数よりも多いポイント数で構成されるポリゴンに属しているポイント全てを選択/解除します。

POINTNDX

引数：リニアなポイントインデックスを表す番号;リニアなポイントインデックスを表す整数値のリストを含む配列;リニアなポイントインデックスを表す整数値を含む初期化ブロック

解説：モデラー内の内部ポイントリストで指定したインデックスの位置に存在するポイントを選択/解除します。

POINTID

引数：MeshDataEditモード関数から得られるポイントID；またはPOINTDX内で記述されるポイントIDの配列または初期化ブロック

解説：指定されるポイントIDで識別されるポイント全てを選択/解除します。`selpoint()`（および`selpolygon()`）を適切に動作させる前に、`selmode()`コマンドを使用してスクリプトをUSERモードへと設定しておく点に注意してください。

プロトタイプ：

```
status selpoint(major[,minor,params]) (CS)
  major    定数; SET もしくは CLEAR
  minor    定数; VOLUME, CONNECT, NPEQ, NPLT, NPGT もしくは POINTID
  params   可変; 前述したマイナーモードに依存
```

selpolygon

`selpolygon()`は`selpoint()`と同類のコマンドであり、ポイントの代わりにポリゴン全体の選択/解除を行います。メジャーモードは`selpoint()`と同じですが、マイナーモードは個数も機能とも異なります。

メジャーモード：

SET: この後にマイナーモードが続くと選択するポリゴンを定義します。

CLEAR: この後にマイナーモードが続くと選択解除するポリゴンを定義します。

マイナーモード：

VOLEXCL

引数：バウンディングボックスの上限/下限を表す二つのベクトル

解説：指定したバウンディングボックス内部に、構成するポイントがすべて完全に含まれているポリゴンを選択/解除します。

VOLINCL

引数：バウンディングボックスの上限/下限を表す二つのベクトル

解説：指定したバウンディングボックス内部に、構成するポイントが少なくとも一つは含まれているポイントを選択/解除します。

CONNECT

引数：なし

解説：既に選択されているポリゴンに連結しているポリゴンを選択します。メジャーモード SET でのみ動作します。

NVEQ

引数：限定するポリゴン内におけるポイント数

解説：引数で指定されたポイント数分だけを含む全ポリゴンを選択/解除します。

NVLT

引数：限定するポリゴン内におけるポイント数

解説：引数で指定されたポイント数よりも少ないポイント数で構成される全ポリゴンを選択/解除します。

NVGT

引数：限定するポリゴン内におけるポイント数

解説：引数で指定されたポイント数よりも多いポイント数で構成される全ポリゴンを選択/解除します。

SURFACE

引数：ポリゴンに割り当てられているサーフェイス名称を表す文字列

解説：指定したサーフェイス名称が割り当てられているポリゴン全てを選択/解除します。

FACE

引数：なし

解説：モデラーにおいて"面"タイプとみなされる全ポリゴンを選択/解除します。

CURVE

引数：なし

解説：モデラーにおいて"カーブ"タイプとみなされる全ポリゴンを選択/解除します。

NONPLANAR

引数：プラナー選択の限度を示すオプション数値

解説：指定した引数よりも少ないプラナーのポリゴンを選択/解除します。オプション引数が省略されている場合、ユーザーのデフォルトプラナーリミットが使用されます。

POLYNDX

引数：リニアなポリゴンインデックスを表す番号；リニアなポリゴンインデックスを表す整数値のリストを含む配列；リニアなポリゴンインデックスを表す整数値を含む初期化ブロック

解説：モデラー内の内部ポリゴンリストで指定したインデックスの位置に存在するポイントを選択/解除します。

POLYID

引数：MeshDataEditモード関数から得られるポリゴンID；またはPOLYNDX内で記述されるポリゴンIDの配列または初期化ブロック

解説：指定されるポリゴンIDで識別されるポイント全てを選択/解除します。

PART

パートの名称で選択します。

BONE

この選択タイプには引数がなく、カレントメッシュ内部にある各タイプのエレメント全てを選択します。これは**Skelegon**と同義語であり、実際Skelegonで指定することも可能です。

SKELEGON PATCH

これらの選択タイプには引数がなく、カレントメッシュ内部にある各タイプのエレメント全てを選択します。

MBALL

この選択タイプには引数がなく、カレントメッシュ内部にある各タイプのエレメント全てを選択します。この場合はMetaballです。`selpoint()`と同様、`selpolygon()`を適切に動作させるためには`selmode()`コマンドを使用してスクリプトをUSERモードへと設定しておいてください。

プロトタイプ：

```
status selpolygon(major [,minor, params]) (CS)
  major    定数; SET もしくは CLEAR
  minor    定数; VOLEXCL, VOLINCL, CONNECT, NVEQ, NVLT, NVGT, SURFACE, FACE,
                 CURVE, NONPLANAR もしくは POLYID
  params   可変; 前述のマイナーモードパラメータに依存
```

selmode

selmode()は様々なモデラー特有コマンド、とりわけselpoint()とselpoolygon()コマンドで使用されます。選択モードにはGLOBAL, USERそれにDIRECTがあります。デフォルトモードはGLOBAL です。

GLOBAL: ユーザーの選択に関係なく全エレメントを選択

USER: ユーザから選択されているエレメントのみ。明示的には何も選択されていない場合、またはボリュームによる選択の場合には、全てのエレメントが選択となります。

DIRECT: ポイントまたはポリゴン選択ツールで直接選択されているエレメント。インターフェイスで現在アクティブになっているモードに関係なく、ポイントとポリゴン双方に適用します。

選択コマンドselpoint()とselpolygon()は、USER選択モードがアクティブでない限りは機能しません。

プロトタイプ：

```
selmode(mode) (IN)
  mode      定数; GLOBAL, USER, もしくは DIRECT
```

selinvert

selinvert()コマンドは現在の選択状態を反転します（つまり選択されているアイテムは選択が解除され、選択されていないアイテムが選択されます）。

プロトタイプ：

```
status selinvert(void) (CS)
```

selhide

selhide()は、SELECTEDオプションが使用されている場合には、選択されているデータ全てをビューから隠し、UNSELECTEDオプションが使用されている場合には、選択されていないデータ全てをビューから隠します。

プロトタイプ：

```
status selhide(which) (CS)
  which     定数; SELECTED もしくは UNSELECTED
```

selunhide

`selunhide()`コマンドは`selhide()`の同類コマンドで、隠されているデータをビューへと戻すために使用されます。

プロトタイプ

```
status selunhide() (CS)
```

第４章：OBJECT AGENTS リファレンス

第5章より先は、以下の **Object Agents** で利用可能なデータメンバとメソッドについて解説していきます。

LightWave アイテム：

第5章： Common（共通）
第6章： Meshes（メッシュ）
第7章： Bones（ボーン）
第8章： Images（イメージ）
第9章： Lights（ライト）
第10章： Cameras（カメラ）
第11章： Scenes（シーン）

共通

第12章： Variables（変数）
第13章： File（ファイル）
第14章： VMap
第15章： Points（ポイント）と Polygons（ポリゴン）
第16章： Controls（コントロール）

レイアウト

第17章： Shader（シェーダー）
第18章： Surface（サーフェイス）
第19章： Texture（テクスチャ）
第20章： TextureLayer（テクスチャレイヤー）
第21章： Image Filter（イメージフィルター）
第22章： Displacement（変位）
第23章： Object Replacement（オブジェクト置き換え）
第24章： Particle（パーティクル）
第25章： Motion（モーション）

第５章：共通データメンバとメソッド

以下のデータメンバとメソッドはレイアウトのアイテム、MESH, BONE, LIGHT, CAMERA それに SCENE に対し共通です。しかし、全てのアイテムがこれらのデータメンバやメソッドで活用可能ではありませんので、注意してください。適用するタイプに依存します。例えば **Light Object** では **Camera Object** の **ZoomFactor** メソッドは使用できません。

データメンバ

name

name にはレイアウトオブジェクトの名称（レイアウトのオブジェクトリストでユーザに表示される名称）を表す文字列が入っています。

filename

filename にはレイアウトオブジェクトのファイル名称（パス含む）を表す文字列が入っています。

parent

parent はレイアウトのオブジェクトの親となるレイアウトオブジェクトの **Object Agent** を指しています（親が存在しない場合には'nil'）。

target

target はレイアウトオブジェクトに対するターゲットとなるレイアウトオブジェクトの **Object Agent** を指しています（ターゲットが存在しない場合には'nil'）。

goal

goal はレイアウトオブジェクトに対するゴールとなるレイアウトオブジェクトの **Object Agent** を指しています（ゴールが存在しない場合には'nil'）。

type

レイアウトオブジェクトの種類（MESH、BONE、LIGHT、CAMERA もしくは SCENE のいずれか）を識別する定数値（読取専用）が入っています。

pointcount

適切なタイプであれば、pointcount はレイアウトオブジェクトのポイント数を返します。適切でない場合（例えばタイプがカメラの場合など）は'nil'となります。

polycount

適切なタイプであれば、polycount はレイアウトオブジェクトのポリゴン数を返します。適切でない場合（例えばタイプがカメラの場合など）は'nil'となります。

shadows

shadows にはレイアウトオブジェクトの現在のシャドーオプションを表す三つのブール値が入っています。

```
[1] == 自己の影がオンの場合には true、オフの場合には false
[2] == 影を落とすがオンの場合には true、オフの場合には false
[3] == 影を受けるがオンの場合には true、オフの場合には false
```

selected, childrenvisible, channelsvisible, locked

selected、childrenvisible、channelsvisible それに locked は、それぞれ指定した状態を表すブール値のフラグです。

id

id にはプロキシとして提供されるレイアウトオブジェクトの整数 ID 値が記入されています。この整数 ID は、シーンファイル内でオブジェクトを一意に識別するためにレイアウトで使用される ID と同一です。

```
generic
{
  obj = Light();
  info(hex(obj.id)); // 表示 "0x20000000"
}
```

genus

`genus` にはオブジェクトの種類が記入されています。種類には `MESH`、`LIGHT`、`CAMERA`、`BONE`、`SCENE` それに `CHANNEL` があります。

visibility

`visibility` には、Scene Object Agent の `visibility()` メソッドからの戻り値の一つが記入されています。

メソッド

param(specifier)

指定子（POSITION や ROTATION など）を指定すると、アイテムの属性に関し以下の情報を得ることが可能になります。

```
getPosition(time), getRight(time), getUp(time), getForward(time),
getRotation(time), getScaling(time), getPivot(time), getWorldPosition(time),
getWorldRotation(time), getWorldRight(time), getWorldUp(time),
getWorldForward(time)
```

これらのメソッドはそれぞれ、指定したタイムインデックスにおいてパラメータに相当する三つの数値を含むベクトルを返します。

firstChannel()

firstChannel()は、オブジェクトにつけられている最初のチャンネルを返します。firstChild()メソッドのように、このメソッドは全チャンネルを通しての反復作業の序文として使用されることになります。利用可能なチャンネルが何もなければ'nil'を返します。

nextChannel()

nextChannel()は、オブジェクトに対するチャンネルリストにある次のチャンネルを返します。利用可能なチャンネルが何もなければ'nil'を返します。

```
light = Light();
c = light.firstChannel();
while(c)
{
  ...
  c = light.nextChannel();
}
```

next()

next()は同じタイプリスト内において次のレイアウトオブジェクト（LScript Object Agent として）を返します。なければ'nil'を返します。

firstChild()

`firstChild()`は、このレイアウトオブジェクトに属している一番最初の子供となるレイアウトオブジェクトの Object Agent を返します（子アイテムが存在しない場合には'`nil`'を返します）。

nextChild()

`nextChild()`は、このレイアウトオブジェクトの次の子供となるレイアウトオブジェクトの Object Agent を返します（存在しない場合には'`nil`'）。

bone()

`bone()`は、このレイアウトオブジェクトに属している最初のボーンであるレイアウトオブジェクトの Object Agent を返します（Object Agent のメソッドである `next()`を使用すると、このレイアウトオブジェクトに割り当てられているレイアウトボーンを全て調べていくことが出来ます）。

limits(state)

`limits(state)`は、ある特定のオブジェクトの状態（`POSITION`, `RIGHT`, `UP`, `FOWARD`, `ROTATION`, `SCALING`, `PIVOTWRIGHT`, `WUP`, `WFORWARD`、または `WPOSITION`）を表す最小値［要素 1-3］と最大値［要素 4-6］を表す要素 6 の数値配列を返します。

isMesh(), isLight(),isCamera(), isBone(), isScene(), isImage(), isChannel(), isEnvelope(), isVMap(), isChannelGroup()

これらのメソッドは Object Agent がプロキシとなっているオブジェクトのタイプを示すブール値 `true`/`false` を返すメソッドです。

setTag(integer, string)

インデックス値と文字列タグの値を指定すると、`setTag()`メソッドはオブジェクトに対しこの文字列を適用します。カレントシーンが保存される時に、タグ情報は関連付けられているオブジェクト共にシーンファイル内に保存されます。

getTag(integer)

インデックス値を指定すると、`getTag()`メソッドはインデックスに対するタグの値を返します。タグが存在しない場合には'`nil`'を返します。

schemaPosition()

`schemaPosition()`はレイアウトのスケマティックビューポートのオブジェクトの位置を表す二つの浮動小数点数値を返します。

server(<class>,[index])

`server（<class>,[index]）`メソッドは、オブジェクトに適用されているアクティブなプラグインの名称を返します。第1引数には照会するプラグインクラスを、第2引数には選択している特定のプラグインに対するインデックス値を指定します。

プラグインクラスには文字列定数（"ItemMotionHandler"のように）か、もしくは以下に示す事前定義の定数値を指定することが出来ます。

```
SERVER_ANIMLOADER_H SERVER_ANIMLOADER_I
SERVER_ANIMSAVER_H SERVER_ANIMSAVER_I
SERVER_CHANNEL_H SERVER_CHANNEL_I
SERVER_CUSTOMOBJ_H SERVER_CUSTOMOBJ_I
SERVER_DISPLACEMENT_H SERVER_DISPLACEMENT_I
SERVER_ENVIRONMENT_H SERVER_ENVIRONMENT_I
SERVER_IMAGEFILTER_H SERVER_IMAGEFILTER_I
SERVER_PIXELFILTER_H SERVER_PIXELFILTER_I
SERVER_FRAMEBUFFER_H SERVER_FRAMEBUFFER_I
SERVER_MASTER_H SERVER_MASTER_I
SERVER_ITEMMOTION_H SERVER_ITEMMOTION_I
SERVER_OBJREPLACEMENT_H SERVER_OBJREPLACEMENT_I
SERVER_SHADER_H SERVER_SHADER_I
SERVER_TEXTURE_H SERVER_TEXTURE_I
SERVER_VOLUMETRIC_H SERVER_VOLUMETRIC_I
```

`_H`はハンドラータイプのサーバー（例えば`IteMotionHandler`）を、`_I` はインターフェイスタイプのサーバー（例えば`ItemMotionInterface`）を返します。

プラグインクラスには、ImageFilterやMasterクラスのようにアイテムを必要としないクラスもあります。これらのクラスタイプはLayout Objectへと提供することが出来る一方で、Scene Object Agentを使用（勿論、このタイプのプラグインはアクティブで）している時のみに正当な値を取得することが出来ます。同様に、オブジェクトに割り当てられているクラスはScene Object Agentでは正当な値を返しません。

第 6 章：MESH OBJECT AGENTS

`Mesh()`コンストラクタは、モデラー、レイアウトどちらの LScript からでも呼び出し可能です。モデラーでは例外的に、現在アクティブなメッシュにはアクセスが可能です。インデックス値ゼロ（0）を使用することで、**Object Agent** はカレントメッシュオブジェクトに対し生成されます。

```
obj = Mesh(); // 最初に読み込まれたオブジェクトからメッシュオブジェクトを作成
obj = Mesh(0); // 現在選択されているオブジェクトからメッシュオブジェクトを作成
obj = Mesh("Cow"); // オブジェクト "cow" からメッシュオブジェクトを作成
```

注意
共通となるデータメンバとメソッド全てが Mesh Object に適用できるわけではないという点に注意してください。

データメンバ

id

`id` には、読み込まれているオブジェクトを一意に識別するためモデラーで内部的に使用されている文字列が入っています。

name

`name` にはインターフェイス上でユーザーから参照できる ID を表すための文字列が入っています。

filename

`filename` にはオブジェクトのフルパス付きファイル名称が入っています。オブジェクトがまだディスクに保存されていない場合、この値は'`nil`'になります。

null

`null` にはオブジェクトがヌルオブジェクトであるかどうかを示すブール値`true`/`false`が入っています。

flags[]

flags[]データメンバは要素5の配列です。最初の三つの要素はshadows[]データメンバの複製値です。残りの二つ、flags[4]とflags[5]はオブジェクトの現在の“カメラ無効”と“光線無効”の状態を示す値が入っています。

メソッド

pointCount([<layer>])

pointCount([<layer>])メソッドは、メッシュオブジェクト内のポイント総数、またはオプション引数を指定することで、個別のレイヤーに存在するメッシュのポイント総数を返します（オブジェクト内の正当なレイヤーを確定するためには'totallayers'データメンバを使用して下さい）。

polygonCount([<layer>])

polygonCount([<layer>])はオブジェクト内部の、またはオプションでレイヤーを指定することにより個別のレイヤー内に存在するポリゴン数を返します。

layer(<point>|<polygon>)

layer(<point>|<polygon>)は、引数で指定したポイントIDまたはポリゴンIDを含む整数レイヤー番号を返します。

position([<point>][<polygon>][<layer>])

position([<point>][<polygon>][<layer>])メソッドは、指定される引数に合わせて二つのタイプの値のうち、どちらか一方を返します。引数がポイントIDの場合にはポイントのベクトル位置を返し、ポリゴンIDであればポリゴンのバウンディングボックスを表す上限ベクトル、下限ベクトルを返します。引数が整数レイヤーIDであればレイヤーにあるメッシュデータのバウンディングボックスlow/highを返します。

最後に引数が何も指定されない場合は、全ての有効なレイヤー内にあるメッシュデータ全体のバウンディングボックスのlow/highベクトルを返します。

vertexCount(<polygon>)

vertexCount(<polygon>)は指定したポリゴンを構成している頂点数を整数値で返します。

vertex(<polygon>,<index>)

`vertex(<polygon>,<index>)`は、指定したポリゴンにおけるインデックスオフセットの位置にあるポイント ID を返します。

select()

`select()`では、Agent がプロキシとなっているオブジェクトを選択状態にします。レイアウトではシーン内でオブジェクトを選択します。モデラーではこの選択したオブジェクトをアクティブな編集状態へと切り替えます。

select(<point>|<polygon>)(モデラーのみ)

`select(<point>|<polygon>)`は指定されたポリゴンまたはポイントを選択します。コンポーネント選択はメッシュ編集独自の選択法なので、このメソッドはモデラーでのみ利用可能です。

例：

この例はレイアウトで動作可能であり、一番最初に読み込まれたオブジェクトから Mesh Object を作成します。その後プロパティをいくつか表示します。

```
@version 2.2
@warnings
@name MeshObjectAgent
@script generic
generic
{
  // 最初に読み込まれたオブジェクトから Mesh Object を作成します
  mesh = Mesh() || error("No Object Loaded!");
  // データメンバ
  info("name: ", mesh.name);
  info("filename: ", mesh.filename);
  // メソッド
  info("points: ", mesh.pointCount(1));
  info("polygons: ", mesh.polygonCount(1));
}
```

第 7 章：BONE OBJECT AGENTS

BONE Objectは、**Mesh Object**の子供であると解釈されます。ボーンはそれ自身だけでは存在できないからです。このため、ボーン情報へアクセスするには、親である Mesh Object を通して行われます。本来 Bone Object は、LScript Mesh Object のデータメンバとなります。

```
mesh = Mesh(); // Mesh Object を作成
bone = mesh.bone(); // 親である Mesh Object から Bone Object を作成
```

注意
共通となるデータメンバとメソッド全てが Bone Object に適用できるわけではないという点に注意してください。

データメンバ

restlength

`restlength` には、ボーンの固定位置（メートル単位）を表す浮動小数点数が入っています。

innerlimit

`innerlimit` には、ボーンの影響範囲限定の開始位置を表す浮動小数点数値が入っています（`flags[2]`が`'true'`の場合にのみ有効）。

outerlimit

`innerlimit` には、ボーンの影響範囲限定の終了位置を表す浮動小数点数値が入っています（`flags[2]`が`'true'`の場合にのみ有効）。

flags[]

`flags[]`には、ボーンの現在のオプションを示す二つのブール値が入っています。

```
[1] == ボーンがアクティブであれば true、そうでない場合には false
[2] == ボーンが影響範囲が限定されていれば true、そうでない場合には false
```

メソッド

restparam(state)

`restparam(state)`からは特別なボーンの状態を表す三つの値の配列が返されます。状態値には定数`POSITION`, `RIGHT`, `UP`, `FORWARD`, `ROTATION`, `SCALING`, `PIVOT`, `WPOSITION`, `WRIGHT`, `WUP`, または`WFORWARD`などが指定できます。

例：

この例では、オブジェクトとその中にあるボーンを指定し、ボーンを次々に調べていき情報を表示します。

```
@version 2.2
@warnings
@name boneObjectAgentTest
@script generic
generic
{
  // フレーム用の変数にフレーム 0 を設定します
  currTime = 0;
  // 一番目のオブジェクトから Mesh Object を作成します
  mesh = Mesh();
  // この mesh Object Agent から最初のボーンに対する Bone Object を作成します
  bone = mesh.bone();
  // ボーンの値が 'nil' になるまで全てのボーンを調べます
  while(bone)
  {
    // bone Object Agent にアクセスすることにより
    // 0 フレームにおけるカレントのボーンを表示します
    info("Bone Name: ", bone);
    // デフォルトの属性
    info("Bone Position: ",
    bone.getPosition(currTime));
    info("Bone Rotation: ",
    bone.getRotation(currTime));
    // 次のボーン移ります
    // ボーンが存在しなければ 'nil' になります
    bone = bone.next();
  }
}
```

第 8 章：IMAGE OBJECT AGENTS

Image Object は、`Image()` コンストラクタを使用して作成されます。LightWave アイテムから作成される大半の **Object Agents** と同様、LightWave に読み込まれているイメージの名称を特定するか（拡張子やパス情報は必要ありません）、引数を指定せずに最初に読み込まれているイメージから Object を作成します。

```
image = Image(); // 最初に読み込まれているイメージから Image Object を作成
image = Image(2); // 二番目に読み込まれているイメージから Image Object を作成
image = Image( "imageName" );// イメージ名称から Image Object を作成
```

全 LightWave アイテムに対し共通となるデータメンバ、メソッドに加え、下記が Image Object Agents 特有のものとして追加されます。

注意
共通となるデータメンバとメソッド全てが Image Object に適用できるわけではないという点に注意してください。

データメンバ

name

`name` にはインターフェイス上に表示されている名称を表す文字列が入っています。

isColor

`isColor` はイメージがカラー画像かグレースケールの画像かを示すブール値を返します。

width

画像の幅（ピクセル単位）を示す値を返します。

height

画像の高さ（ピクセル単位）を示す値を返します。

hasAlpha

`hasAlpha` は、画像にアルファデータが含まれているかを示すブール値のフラグを返します。このフラグが `true` であれば、`alpha()` と `alphaSpot()` を使用してアルファデータにアクセスすることが可能です。

メソッド

filename(frame)

`filename(frame)`は、指定したフレームにおける Image Object Agent のパス情報を返します。

luma(x, y)

`luma(x, y)`は、指定する整数座標値（x,y）におけるピクセルのグレースケール値を返します。

rgb(x, y)

`rgb(x, y)`は、指定する整数座標値（x,y）におけるピクセルの RGB 値を返します。

alpha(x, y)

`alpha(x, y)`は、指定する整数座標値（x,y）におけるピクセルのアルファ値を返します。アルファチャンネルのデータには、一組の RGB の値に対し一個の数値が含まれています。

例：

この例では、最初に読み込まれたイメージから Image Object を作成し、プロパティをいくつか表示します。

```
@version 2.2
@warnings
@script generic
@name ImageObjectAgentTest
generic
{
  // 最初に読み込まれたイメージから Image Object Agent を作成
  image = Image() || error("No images loaded!");
  // データメンバの表示
  info("Name: ", image.name);
  info("Width: ", image.width);
  info("Height: ", image.height);
  info("isColor: ", image.isColor);
  info("hasAlpha: ", image.hasAlpha);
}
```

第 9 章：LIGHT OBJECT AGENTS

Light Objects は、`Light()` コンストラクタを使用して作成されます。スクリプトはライトの名称を渡すか、またはライトのインデックス番号を使用することで、使用しているライトを識別することが可能です。引数が何も渡されない場合には、最初のライトが使用されます。

```
lgt = Light(); // 最初のライトから Light Object を作成
lgt = Light(2); // 二番目のライトから Light Object を作成
lgt = Light( "Light1" ); // 名称 "Light1" のライトから Light Object を作成
```

注意
共通となるデータメンバとメソッド全てが Light Object に適用できるわけではないという点に注意してください。

データメンバ

type

`type` には、このライトの種類を識別する定数整数値（読み込み専用）が入っています。（`DISTANTLIGHT`、`POINTLIGHT`、`SPOTLIGHT`、`LINEARLIGHT` もしくは `AREALIGHT` のいずれか）

shadowtype

`shadowtype` には、ライト種を識別する定数整数値（読み込み専用）が入っています。（`SHADOWRAYTRACE`、`SHADOWMAP` もしくは `SHADOWOFF` のいずれか）

coneangles[]

ライトが `SPOTLIGHT` の場合、`coneangles[]` には、このライトに対するコーン角度を表す二つの浮動小数点数値の配列が入っています。最初の要素[1]には全体のスポットライトのスポットコーン角度の半分の値が、要素[2]にはスポットライトのスポットソフト角度の幅が入っています。

range

フォールオフが設定されており、ライト種が `DISTANTLIGHT` 以外の場合に、`range` には明るさのフォールオフの最大範囲（メートル単位）が入っています。

flags[]

`flags[]`データメンバは要素6の配列であり、各要素はSDK関数から返されるフラグの値に対応しています。詳細に言うと、各要素はライトに関する以下の情報を提供します。

`flags[1]` 範囲限定が有効の場合には `true` を返します
`flags[2]` 拡散レベル有効の場合には `true` を返します
`flags[3]` 反射光有効が有効の場合には `true` を返します
`flags[4]` コースティクス有効が有効の場合には `true` を返します
`flags[5]` レンズフレア有効が有効の場合には `true` を返します
`flags[6]` ヴォリュームライトが有効の場合には `true` を返します

メソッド

ambient(time), rgbambient(time)

`ambient(time)`, `rgbambient(time)`は、指定した`time`インデックスにおける全照明の環境光の色を表すベクトル、または三つの値を保持する配列を返します。`ambient()`はチャンネルの値をベクトルとして返します。ベクトルの個々の要素はパーセンテージで、範囲0.0～1.0となります。一方、`rgbambient()`はチャンネルの値を三つの整数値として返し、範囲は0～255となります。

color(time), rgbcolor(time)

`color(time)`、`rgbcolor(time)`は、指定した`time`インデックスにおけるライトの色を表すベクトル、もしくは要素3の配列を返します。`color()`はチャンネルの値をベクトルとして返します。ベクトルの個々の要素はパーセンテージで、範囲0.0～1.0となります。一方、`rgbcolor()`はチャンネルの値を三つの整数値として返し、範囲は0～255となります。

例：

この例では、シーン内の一番初めのライトからLight Objectを作成します。その後Light Objectのデータメンバとメソッドから保持されている値をいくつか表示します。

```
@version 2.2
@warnings
@name LightObjectAgentTest
@script generic
// 大域変数の設定
frame = 0;
generic
{
  light = Light();
  // データメンバを表示
```

```
  info("Name: ", light.name);
  info("Type: ", light.type);
  // メソッドを表示
  info("Position: ", light.getPosition(frame));
  info("Rotation: ", light.getRotation(frame));
  info("Ambient: ", light.ambient(frame));
  info("Color: ", light.color(frame));
}
```

第 10 章：CAMERA OBJECT AGENTS

Camera Objectsは、Camera()コンストラクタを使用して作成されます。このコンストラクタに何も引数を指定しない場合には、シーン中の一番最初のカメラからCamera Objectを作成します。引数nameには、使用されている特定のカメラの名称を渡すことが可能です。

```
camera = Camera();  // 1番目のカメラから Camera Object を作成
camera = Camera(3); // 3番目のカメラから Camera Object を作成
camera = Camera( "camera1" ); // "camera1" という名称をもつカメラから
                              // Camera Object を作成
```

注意
共通となるデータメンバとメソッド全てがCamera Objectに適用できるわけではないという点に注意してください。

メソッド

zoomFactor(time)

zoomFactor(time)は、指定した'time'インデックスにおけるカメラのズームファクターの値を表す浮動小数点数値を返します。

focalLength(time)

focalLength(time)は、指定した'time'インデックスにおけるカメラの焦点の長さを表す浮動小数点数値を返します。

focalDistance(time)

focalDistance(time)は、指定した'time'インデックスにおけるカメラの焦点距離（メートル単位）を表す浮動小数点数値を返します。

fStop(time)

fStop(time)は、指定した'time'インデックスにおける開放F値を表す浮動小数点数値を返します。

blurLength(time)

`blurLength(time)`は、指定した'`time`'インデックスにおけるブラーの強さ（メートル単位）を表す浮動小数点数値を返します。

fovAngles(time)

`fovAngle(time)`は、指定した'`time`'インデックスにおけるカメラの視野角を表す浮動小数点数値を返します。要素[1]には水平角、要素[2]には垂直角が入っています。これらの角度（ラジアン単位）はカメラの方向を中心とした角度です。

例：

この例ではCamera Objectを作成し、プロパティをいくつか表示しています。

```
@version 2.2
@warnings
@name CameraObjectAgent
@script generic
// 大域変数の設定
frame = 0;
generic
{
  camera = Camera();
  // データメンバを表示
  info("Name: ", camera.name);
  // メソッドを表示
  info("Position: ", camera.getPosition(frame));
  info("Rotation: ", camera.getRotation(frame));
  info("zoomFactor: ", camera.zoomFactor(frame));
  info("focalDistance: ", camera.focalDistance(frame));
  info("fStop: ", camera.fStop(frame));
}
```

第１１章：SCENE OBJECT AGENTS

Object Agent コンストラクタである Scene() は、カレントシーンのデータメンバやプロパティを扱うメソッドを含む Object を作成します。LightWave アイテムを扱う大半の Object Agents とは異なり、Scene() コンストラクタは引数を持ちません。

```
scene = Scene(); // カレントシーンから Scene Object を作成
```

注意
共通となるデータメンバとメソッド全てが Scene Object に適用できるわけではないという点に注意してください。

データメンバ

backdroptype

backdroptype には、シーンにおける現在の背景タイプの値が入っています。SOLID または GRADIENT となります。

compfg, compbg, and compfgalpha

compfg、compbg、それに compfgalph は、シーンのカレントの合成状態に関する情報を提供します。これらの値には全てアクティブな合成画像（それぞれ前景、背景、前景アルファ）に対する Image Object Agent が含まれています。合成する画像がアクティブでなければ 'nil' が入ります。

currenttime

currenttime には、レイアウトインターフェイス上で現在選択されているタイムインデックス値が入っています。

dynamicupdate

dynamicupdate は、レイアウトの動的な更新の設定状態に関する情報を提供します。この情報はデータメンバ内で利用可能であり、以下の値のうちの一つをとります。

```
DYNUP_OFF
DYNUP_DELAYED
DYNUP_INTERACTIVE
```

displayopts[]

displayopts[]は、レイアウトにおける表示関連の情報を提供します。配列の各要素は以下のアイテムについての情報です。

displayopts[1] モーションパス表示がオンの場合はブール値'true'
displayopts[2] ハンドル表示がオンの場合はブール値'true'
displayopts[3] IKチェイン表示がオンの場合はブール値'true'
displayopts[4] パッチ外郭表示がオンの場合はブール値'true'
displayopts[5] セーフエリア表示がオンの場合はブール値'true'
displayopts[6] フィールドチャート表示がオンの場合はブール値'true'

filename

filenameには、シーンのファイル名称を表す文字列が入っています。

fogtype

fogtypeには、シーンにおけるアクティブなフォグのタイプを表す値が入っています。NONE、LINEAR、NONLINEAR1もしくはNONLINEAR2となります。

fps

fpsには、シーンに対するフレーム/秒（デフォルトは30）を表す整数値が入っています。

framestart, renderstart

framestart、renderstartには、レンダリングの開始フレームを表す整数値が入っています。

framestep, renderstep

framestep、renderstepには、フレームステップを表す整数値が入っています。

frameend, renderend

frameend、renderendには、レンダリングの終了フレームを表す整数値が入っています。

frameheight

frameheightには、レンダリングされるフレームの高さを表す値が入っています。

framewidth

framewidthには、レンダリングされるフレームの幅を表す値が入っています。

generalopts[]

`generalopts[]`には、レイアウトの一般的な設定に関わる情報が入っています。以下、配列の特定の要素についての説明です。

```
generalopts[1] ツールバー非表示が有効であればブール値 'true'
generalopts[2] ツールバー位置右側が有効であればブール値 'true'
generalopts[3] その場でペアレントが有効であればブール値 'true'
generalopts[4] 端数フレームを許可が有効であればブール値 'true'
generalopts[5] スライダにキー表示が有効であればブール値 'true'
generalopts[6] 実レートで再生が有効であればブール値 'true'
```

limitedregion[]

`limitedregion[]`には、シーンの限定範囲の位置を表す要素4の整数配列が入っています。

```
[1] == x0
[2] == y0
[3] == x1
[4] == y1
```

minsamplesperpixel

`minsamplesperpixel`には，現在のレンダリングオプションに基づいて、最終画像内において1ピクセルについての最小サンプル数を表す整数値が入っています。

maxsamplesperpixel

`maxsamplesperpixel`には、現在のレンダリングオプションに基づいて、最終画像内において1ピクセルについての最大サンプル数を表す整数値が入っています。

name

シーンの名称をあらわす文字列が入っています。

pixelaspect

シーンのピクセルアスペクト比（ピクセルの幅/ピクセルの高さ）を表す浮動小数点数値が入っています。

previewstart, previewend, previewstep

`previewstart`、`previewend`、それに`previewstep`はLightWaveのプレビューシステムにより使用される値（ユーザーインターフェイス上の編集フィールドに一致する値）を表しています。

recursiondepth

recursiondepth には、レンダリング時に使用される反射回数の上限を表す整数値が入っています。

rendertype

rendertype には、シーンに設定されているレンダリングの種類を識別する定数整数値（読取専用）が入っています（WIRERENDER、QUICKRENDER または REALISTICRENDER）。

renderopts[]

renderopts[]には、シーンに関するカレントオプションを表す要素 8 のブール配列が入っています。

[1] 影トレースがアクティブな場合は true
[2] 反射トレースがアクティブな場合は true
[3] 屈折トレースがアクティブな場合は true
[4] フィールドレンダリングがアクティブな場合は true
[5] リバースフィールドレンダリングがアクティブな場合は true
[6] モーションブラーがアクティブな場合は true
[7] 被写界深度がアクティブな場合は true
[8] 範囲限定がアクティブな場合は true

totalpoints

totalpoints には、シーン中のメッシュポイントのポイント総数を表す整数値が入っています。

totalpolygons

totalpolygons には、シーン中のメッシュポイントのポリゴン総数を表す整数値が入っています。

メソッド

backdropRay(time,ray)

`backdropRay(time,ray)`メソッドは、ある特定の時刻において指定されたレイを交差する色をベクトルとして返します。引数`ray`には正規化ベクトルを指定してください。

backdropColor(time)

`backdropColor(time)`メソッドは、指定された時刻における空の色、水平の色、地平の色、地面の色をベクトルで返します。

backdropSqueeze(time)

`backdropSqueeze(time)`は、指定した時刻における水平の割合と地平の割合を浮動小数点数値で返します。

firstSelect(), nextSelect()

`firstSelect()`と`nextSelect()`は、カレントシーンで選択されているオブジェクトを循環するのに使用されます。メソッドはLayout Object Agentにおける`firstChild()`や`nextChild()`と同様、他の反復メソッドに相当する形式で使用されます。

この二つのメソッドは動的なものです。現在の選択が呼び出しの間で修正されれば、予測通りの動作ではなくなります。`getSelect()`メソッドは現在選択されているオブジェクトを全て返し、選択修正コマンドが使用されたかどうかを返します。

fogMinDist(time)

`fogMinDis(time)`メソッドは、視点（通常はカメラ）からフォグ効果の最短距離を返します。

fogMaxDist(time)

`fogMaxDis(time)`メソッドは、視点（通常はカメラ）からフォグ効果の最長距離を返します。

fogMinAmount(time)

`fogMinAmount(time)`は、フォグの最小量（最短距離における量）を返します。フォグ量の範囲は0.0～1.0です。

fogMaxAmount(time)

`fogMaxAmount(time)`は、フォグの最大量（最長距離における量）を返します。フォグ量の範囲は0.0～1.0です。

fogColor(time)

`fogColor(time)`メソッドは、指定された時刻におけるフォグの色をベクトルとして返します。

getSelect()

`getSelect()`メソッドは、シーン内で現在選択されているアイテムを全て、Object Agentsのリストとして返します。`getSelect()`が引数無しで呼び出される場合、選択されたアイテムがメッシュ、ボーン、ライト、カメラのいずれの種類にかかわらず、全アイテムを返します。フラグの値によって、特定の種類に対して選択を限定することが可能です。フラグには`MESH`、`BONE`、`LIGHT`、そして`CAMERA`を指定することが出来ます。

`firstSelect()`と`nextSelect()`メソッドもまだ特別な処理が必要となる場合には利用価値がありますが、このメソッドを使用することで、選択されているアイテムをより簡単に確定できるようになります。

renderCamera(time)

時刻を指定することで、`renderCamera(time)`メソッドはその時刻におけるシーン内でアクティブなカメラをCamera Object Agentとして返します。

schemaPosition()

`schemaPisition()`は、レイアウトのスケマティックビュー内においてオブジェクトエントリのXY座標値を取得します。名称またはプロキシを指定すると、このメソッドはこの位置を示す二つの浮動小数点数値を返します。

CommandSequenceの関数である`SchematicPosition()`と一緒に利用すれば、スクリプトはレイアウトのスケマティックビューポートの完全な修正を管理することが出来ます。

visibility()

visibility()は、シーン内のオブジェクトの表示状態を決定します。既存オブジェクトの名称もしくはオブジェクトのプロキシを指定すると、このメソッドは以下の指定子の一つを返します。

```
VIS_VISIBLE
VIS_HIDDEN
VIS_BOUNDINGBOX
VIS_VERTICES
VIS_WIREFRAME
VIS_FFWIREFRAME
VIS_SHADED
VIS_TEXTURED
```

例：

この例ではScene Objectを作成し、firstSelect()とnextSelect()メソッドを使用して、選択されているアイテムを循環させます

```
@version 2.2
@warnings
@name SceneObjectAgent
@script generic
// 大域変数の設定
frame = 1;
generic
{
  scene = Scene();
  // データメンバの表示
  info("Filename: ", scene.filename);
  info("Total Points: ", scene.totalpoints);
  info("Total Polygons: ", scene.totalpolygons);
  // メソッドの表示
  info("RenderCamera: ", scene.renderCamera(frame));
  info("BackdropColor: ", scene.backdropColor(frame));
  // firstSelect() と nextSelect() メソッドを使用して
  // オブジェクトを循環します
  obj = scene.firstSelect();
  while(obj)
  {
    info(obj);
    obj = scene.nextSelect();
  }
}
```

第 12 章：VARIABLE MESSAGES

整数データタイプ（整数、番号、文字列など）は以下のメソッド群に応答します。

メソッド

size()

`size()`はアイテムのサイズを返します。

`<integer>`は値自身を返します。
`<number>`は値自身を返します。
`<string>`は文字数を返します。
`<vector>`は1を返します。
`<array>`は割り当てられている総要素数を返します。

count()

`count()`はアイテム数を返します。

`<integer>`は1を返します。
`<number>`は1を返します。
`<vector>`は1を返します。
`<string>`は文字数を返します。
`<array>`は値を含むよう素数を返します。

asInt()

`asInt()`は整数として解釈されるデータを返します。

asNum()

`asNum()`は番号として解釈されるデータを返します。

asStr()

asStr()は文字列として解釈されるデータを返します。

asVec()

asVec()はベクトルとして解釈されるデータを返します。

pack()

pack()はデータを圧縮し、処理が成功したかどうかをブール値true/falseで返します。

<integer>は何もしません。
<number>は何もしません。
<vector>は何もしません。
<string>は先行ホワイトスペースを除去します。
<array>は要素を圧縮し、間に挟まっている'nil'の値を取り除きます。

trunc()

trunc()はデータを切り捨て、処理が成功どうかを示すブール値true/falseを返します。

<integer>は何もしません。
<number>は何もしません。
<vector>は何もしません。
<string>は先行ホワイトスペースを除去します。
<array>は値'nil'を持つ要素を切り捨て、取り除くことで、配列のサイズを変えます。

sortA()

sortA()は昇順にデータを並び替え、処理が成功すればtrueを、失敗すればfalseを返します。

<integer>は何もしません。
<number>は何もしません。
<vector>は何もしません。
<string>は昇順に文字を並び替えます。
<array>は圧縮をかけた後、昇順に要素を並び替えます。整数、番号、文字列は並び替え可能です。

sortD()

`sortD()`は降順にデータを並び替え、処理が成功すれば`true`を、失敗すれば`false`を返します。

`<integer>`は何もしません。

`<number>`は何もしません。

`<vector>`は何もしません。

`<string>`は降順に文字を並び替えます。

`<array>`は圧縮をかけた後、降順に要素を並び替えます。整数、番号、文字列は並び替え可能です。

isNil()

`isNil()`は、データタイプが'`nil`' であるかを判定し、ブール値`true`または`false`を返します。機能的には等価テスト（つまり"`data == nil`"）と同じ動作を行います。

isInt()

`isInt()`は、データタイプが整数であるかを判定し、ブール値`true`もしくは`false`を返します。これは等価判定ではありません。

isNum()

`isNum()`は、データタイプが数値であるかを判定し、ブール値`true`もしくは`false`を返します。これは等価判定ではありません。

isStr()

`isStr()`は、データタイプが文字列であるかを判定し、ブール値`true`もしくは`false`を返します。これは等価判定ではありません。

isVec()

`isVec()`は、データタイプがベクトルであるかを判定し、ブール値`true`もしくは`false`を返します。これは等価判定ではありません。

reduce()

reduce()メソッドは、データタイプから重複値を除去します（主として文字列や配列に対し便利な機能です）。重複値は線形順、つまり他方に対し昇順になっていると予測されます。reduce()を呼び出す前に、並び替えメソッドの一つを実行しておくべきです。

以下はreduce()メソッドの例です。線形となっているポリゴンIDは配列内に格納され、重複が起こる可能性が高い状況にあります。sortA()とreduce()を呼び出すことで、結果配列から重複エントリを除去します。

```
...
for(x = 1;x <= pointCnt;x++)
{
  rawPolys = points[x].polygon();
  polyCnt = rawPolys.count();
  for(y = 1;y <= polyCnt;y++)
  polySelect += rawPolys[y];
}
polySelect.sortA();
polySelect.reduce();
...
```

例：

以下のコードの断片では、要素8の配列を作成し、ランダムに値を入れ込みます。pack()とtrunc()関数が、この配列に対し処理を行います。

```
q[8] = nil;
q[2] = 1.5; q[4] = "Bob"; q[5] = <1,2,3>;
info(q.size());
info(q.count());
info(q);
q.pack();
info(q);
q.trunc();
info(q);
```

このスクリプトコードは以下のようなメッセージを出します。

```
8
3
(nil) 1.5 (nil) "Bob" <1,2,3> (nil) (nil) (nil)
1.5 "Bob" <1,2,3> (nil) (nil) (nil) (nil) (nil)
1.5 "Bob" <1,2,3>
```

全てのデータタイプが各メッセージに対応している間、全ての操作に意味があるわけではありません。例えば、整数の圧縮や切捨てなどは何の処理も行われません。

第 13 章：FILE OBJECT AGENTS

LScript の **File Object Agent** を使用すると、OS のディスクファイル管理機能にアクセスできます。LScript のコンストラクタ `File()` は、このタイプのオブジェクトを返します。いったん変数にアサインされてしまえば、以下の File Object Agent メソッドがホスト変数を介してアクセス可能になります。

```
// リストされているファイルから File Object を作成
// (読み込みモード専用)
file = File( "c:/lightwave/scenes/tmp.lws" );
```

メソッド

open(string, mode)

`open(string, mode)` は、指定したモード（読み込みには `"r"`、書き出しには `"w"`）で新規ファイルを開きます。新たに開かれたファイルは、既存の **File Object Agent** に割り当てられます。

IsOpen()

`IsOpen()` は、ファイルが開いておりアクセス可能かどうかを示すブール値を返します。

reopen(mode)

`reopen(mode)` は、現在開いているファイルを閉じ、指定したモードでもう一度開きなおします（読み込みには `"r"`、書き出しには `"w"`）。

close()

`close()` は、現在開かれているファイルを閉じます。

eof()

`eof()` は、ファイルがファイル終端まで到達したかどうかを示すブール値（`true`/`false`）を返します。

rewind()

rewind()は、読み込み/書き込みファイルをファイル先端までリセットします。

name()

name()は、現在開かれているファイルのファイル名称（本来指定されている場合には名称とパス）を指す文字列を返します。

write(dataType[,itemN]), writeln(dataType[,itemN])

write()は、ファイルに対し単一もしくは複数の引数を配置します。writeln()はwrite()と同じような関数ですが、行に改行文字を付加します。ファイルをバイナリモードで開いている場合には、これらのコマンドは無効なコマンドとなります。

read()

read()は、テキスト行を文字列としてファイルから返します。ファイルをバイナリモードで開いている場合には、これらのコマンドは無効なコマンドとなります。

readNumber()

アスキーモードでは、readNumber()関数はファイルから一連の文字を読み込み、単一数値（浮動小数点数）へと変換します。バイナリモードでは、単一数値（サイズはdouble）がファイルから読み込まれます。

readVector()

アスキーモードでは、readVector()関数はファイルから一連の文字列を読み込み、ベクトルタイプのデータとして返します。ファイル内のデータはスペースで区切られた状態の浮動小数点数として存在します。バイナリモードでは、サイズが3個の数値（サイズはdouble）が読み込まれベクトルとして返されます。

parse(string)

parse(string)はファイルから1行読み込み、行のトークンとして表されている変数の要素数を返します。これらの要素は文字列配列に保存されます。トークンはキャラクタ文字列引数として指定されている文字列で区切られます。この関数はバイナリモードでは無効です。

nl()

nl()はファイルに新規行を書き込みます。この関数はバイナリモードでは無効です。

linecount()

ファイルがアスキーモードで開かれているときには、`linecount()`関数は現在開かれているファイルからテキスト行の総数を、バイナリモードではファイルのサイズをバイト数として返します。

line([integer])

アスキーモードにおける`line([integer])`関数は、引数無しの場合、ファイルにおける現在の行番号を返します。ファイル内でテキスト行の範囲内にある整数値が渡された場合には、指定された行番号へとファイルポインタを移動します。バイナリモードでは無効です。

readInt()

アスキーモードにおける`readInt()`関数は、テキストファイルから文字列群を読み取り、整数タイプのデータを返します。バイナリモードでは単一のバイナリの値（サイズは整数）がファイルから読み込まれます。

readByte()

アスキーモードにおける`readBytes()`関数は、テキストファイルから単一文字を整数値として返します。バイナリモードでは、バイナリの値（サイズは符号なし文字）をファイルから読み込み、整数値として返します。

writeNumber(number)

アスキーモードにおける`writeNumber(number)`関数は、テキストファイルに値を浮動小数点数値の文字形式で書き込みます。バイナリモードでは、単一のバイナリの値（サイズはdouble）をファイルに書き込みます。ファイルに書き込まれたバイト数を返します。

writeInt(integer)

アスキーモードでは、`writeInt(number)`関数はテキストファイルに値を整数の文字形式で書き込みます。バイナリモードでは、単一のバイナリの値（サイズは整数）をファイルに書き込みます。ファイルに書き込まれたバイト数を返します。

writeByte(integer)

アスキーモードでは、`writeByte(number)`関数はテキストファイルに値を整数の文字形式で書き込みます。バイナリモードでは、単一のバイナリの値（サイズは符号なし文字）をファイルに書き込みます。ファイルに書き込まれたバイト数を返します。

offset([integer[,method]])

パラメータ無しに呼び出されると、offset([integer[,method]])メソッドはファイルの現在のバイトオフセット値を返します。整数値を指定するとoffsetメソッドに従って、ファイル内の指定したオフセットの箇所にファイルポインタを配置します。このオプションであるオフセットメソッドは以下の値になります。

FROMSTARTはファイル始端からの位置（メソッドが指定されていない場合にはこれがデフォルトとなります）

FROMENDはファイル終端からの位置

FROMHEREは指定されたオフセット値は現在のポインタ位置からの相対位置

例：

```
@version 2.2
@warnings
@script generic
@name FileObjectAgentTest
// 大域変数はここに
generic
{
  file = "C:/newtek/scenes/benchmark/DOF.lws";
  f = File(file,"r") || error("ファイルが開けません'",file,"'");
  if(f.linecount())
  {
    // メソッドの値を表示
    info("Linecount: ", f.linecount());
    // 開始行を読み込んで表示
    line1 = f.read();
    if(line1 != "LWSC")
      error("有効なLightWave Sceneではありません!");
    else
      info("line1: ", line1);
    // 次の行を読み込んで表示
    line2 = f.read();
    info("line2: ", line2);
  }
}
```

第 14 章：VERTEX MAP OBJECT AGENTS

VMap Object Agents用のコンストラクタは`VMap()`であり、大半の場合グローバルで使用されます。

```
vmap = VMap(<VMap Type>);
```

このコンストラクタは**VMap Object Agent**を返します。VMapは特定のVMap タイプの名称か整数インデックス値によって識別が可能です。これらのVMapの種類は以下に記すとおりです。

```
VMSELECT "select"
VMWEIGHT "weight"
VMSUBPATCH "subpatch"
VMTEXTURE "texture"
VMMORPH "morph"
VMSPOT "spot"
VMRGB "rgb"
VMRGBA "rgba"
```

`VMap()`は、指定したタイプの頂点マップがシステムに存在しない場合には'`nil`'を返します。特定の頂点マップタイプを引数で指定せずに`VMap()`を呼び出すと、システム内で最初に遭遇する頂点マップを返します。この最初の頂点マップから`next()`メソッドを使用して、他の全ての頂点マップを通して繰り返すことが可能です。`next()`メソッドは頂点マップの種類の境界を無視しますので、特定のタイプの頂点マップだけを処理したい場合には、種類を注意して見ていかなくてはなりません。

VMap Object Agentはまた、モデラー内部でユーザー定義の頂点マップを作成することが可能です。使用されるメソッドは、事前定義タイプの頂点マップを新規作成するメソッドとかなり似ています。カスタムの頂点マップを作成するためには、事前定義タイプのIDの代わりにカスタムマップタイプを提供する必要があります。カスタムマップタイプは一意の4文字シーケンスで初期化ブロック内に置いておきます。例えば、1ポイントにつき二つのデータエレメントを持つNMBKという種類の頂点マップ`my_normal`を作成するためには、以下のようなコードを使用します。

```
...
@define VMAP_NMBK @'N','M','B','K'@
...
normMap = VMap(VMAP_NMBK,"my_normals",2);
```

カスタム頂点マップは、アプリケーションのインターフェイスにはどこにも出てきません。この種類はアプリケーションに認知されているわけではないので、カスタム頂点マップに対する割り当てが出来ないのです。ただし、保存時にはオブジェクトファイル内に保存され、オブジェクトファイル内部で保存され続けます。

VMap()コンストラクタは、モードに関係なく（カスタムであろうと事前定義であろうと）新規頂点マップの生成に失敗した場合には'nil'を返します。

データメンバ

name

name は **Object Agent** に割り当てられている頂点マップの名称です。

dimensions

dimensions は、マップ内の各頂点に割り当てられている値の数を表します。この値はゼロ（0）でも構いません。

type

type は **Object Agent** に割り当てられている頂点マップの種類です。上記定義の定数値の一つとなります（VMSELECT、VMWEIGHT など）。

メソッド

count()

count()は、現在の環境内にある **Object Agent** タイプの頂点マップの総数を返します。

isMapped(<point>)

isMapped(<point>)は、マップ内に指定したポイント ID が存在するかどうかを返します。戻り値は true もしくは false です。

getValue(<point>[,<index>])

`getValue(<point>[,<index>])`は、指定したポイントIDに割り当てられている頂点マップの値を返します。インデックス値が指定されていない場合、返されるアイテム数は'dimensions'データメンバの値と等しくなります。ポイントが頂点マップに存在しない場合、また頂点マップの次元数がゼロ（0）の場合には、'nil'が返されます（`isMapped()`メソッドと/もしくは'dimension'データメンバを使用して、このような事態は出来るだけ避けるようにして下さい）。

setValue(<point>,<value>|<array>[,<index>])（モデラーのみ）

`setValue(<point>,<value>|<array>[,<index>])`は、頂点マップ内の特定のポイントIDに値を設定します。個々の値が提供されている場合には、その値はポイントに対する一番目の値スロットに置き換わります。オプションであるインデックス値は、個別の値をポイントの値スロットへと配置します。

また、ポイントIDへの値として配列値も指定可能です。オプションインデックスが含まれていない場合には、配列内の各要素は頂点マップ内のポイントID用の一致する値スロットへと置き換わります。インデックスが指定されている場合には、配列内の要素は指定したインデックスオフセットからのポイントID用の値へと配置されます。

`setValue()`メソッドを使用した頂点マップの修正はメッシュ編集であるとみなされ、モデラーLScript内部でのみ実行が可能になります。従って、このメソッドはレイアウトLScriptから生成された**Object Agents**には存在しません。さらにこのメソッドの呼び出しを成功させる前に**Mesh DataEdit**セッションの起動内部で行わなければなりません。

例：

以下は、ユーザーに既存の**Weight VMap**を選択させるモデラーのLScriptの例です。それぞれの値に対し、均等に拡大縮小を行っています。

```
@version 2.1
@warnings
main
{
  vmap = VMap(VMWEIGHT) || error("メッシュ内部にWeight Mapはありません!");
  while(vmap && vmap.type == VMWEIGHT)
  {
    vmapnames += vmap.name;
    vmap = vmap.next();
  }
  reqbegin("Scale Weight VMap",true);
    c1 = ctlpopup("VMap",1,vmapnames);
    c2 = ctlnumber("Scale by (%)",50.0);
```

```
        return if !reqpost();
        vndx = getvalue(c1);
        amount = getvalue(c2) / 100.0;
      reqend();
      vmap = VMap(vmapnames[vndx]) || error("Could not instance VMap
'",vmapnames[vndx],"'!");
      selmode(USER);
      moninit(editbegin());
      foreach(p,points)
      {
        if(vmap.isMapped(p))
        {
          values = vmap.getValue(p);
          for(x = 1;x <=
          vmap.dimensions;x++)
          values[x] *= amount;
          vmap.setValue(p,values);
        }
        monstep();
      }
      monend();
      editend();
    }
```

第 15 章 : POINT と POLYGONS OBJECT AGENTS

Point と **Polygon** のデータタイプには、独自のメンバとメソッドの組み合わせが与えられています。どちらの場合でも、LScript 内において他からはアクセスが出来ない追加情報が、これらのインターフェイスを通して利用可能になります。

Point と **Polygon** メッセージは、メッシュデータセッションの間でのみ有効です（つまり `editBegin()` と `editEnd()` の間）。

Point

Point データタイプは以下のメッセージに応答します。

polygon/polygon()

Point データタイプの `polygon/polygon()` エレメントは、コンテキストに基づく異なるデータを返すのは負担がかかります。エレメントが（データメンバとして）直接アクセスされる場合、ポイントが属しているポリゴンの番号が返されます。メソッド `context` が呼び出される場合には、単数もしくは複数のポリゴン ID が呼び出し側に返されます。ポイントがどのポリゴンにも属していない場合には、データメンバ `context` はゼロ（0）を返し、このメソッドコンテキストは'`nil`'を返します。データメンバは読取専用です。

x, y, z

`x,y,z` の値は指定した軸に沿ったポイントの位置を返します。この値は読み込むことも、ポイント位置を代入して修正することも可能です。

Polygon

データタイプ **Polygon** は以下のメッセージに応答します。

surface

`surface` は、現在ポリゴンに割り当てられているサーフェイスの名称です。このデータメンバは読み取ることも値を割り当てることも可能です。割り当てるサーフェイス名称が存在しない場合は、まずLScriptはそのサーフェイスを作成します。

pointCount

`pointCount` は、ポリゴンを構成するポイント数を表す整数値です。このデータメンバは読取専用です。

layer

`layer` は、ポリゴンが存在するモデラーのレイヤーを指す整数値です。このデータメンバは読取専用です。

points[]

`points[]`は、ポリゴンを構成するポイント全てを含むポイントIDの配列です。ここには'`pointCount`'要素が含まれています。このデータメンバに含まれてる要素は読取専用です。

isCurve()

`isCurve()`は、ポリゴンがカーブ（`true`）か面（`false`）かを示すブール値です。

hasCCEnd()

ポリゴンがカーブの場合、`hasCCEnd()`メソッドは終端が制御点であれば `true` を返します。

hasCCStart()

ポリゴンがカーブの場合、`hasCCStart()`メソッドは始端が制御点であれば `true` を返します。

setPoints()

setPoints()メソッドはポリゴンを構成するポイントを設定することが出来ます。事前定義しているポイントIDを含む参照配列、事前定義ポイントIDのリスト、または単数または複数のベクトル値セットを受け取ります。ベクトル値は最初にまずポイントへと変換され、それからポリゴンへと適用されます。

第 16 章：CONTROL OBJECT AGENTS

ctlstring()やctlnumber()などの関数から返される **Requester** コントロール ID の値で、**Control** データタイプに独自のオブジェクト属性を設定することが可能になります。

データメンバ

value

value データメンバにはコントロールで表示されるコンテキストの値が入っています。この値の読み込みや書き込みは、それぞれコントロール ID への setvalue()や getvalue()関数を使用する処理と同様です。

active（読取専用）

active は、コントロールの現在のアクティブ状況を示すブール値を返します。'true' であれば現在コントロールは有効であり、ユーザーと対話可能であることを示しています。

visible（読取専用）

visible はコントロールの現在の可視状況を示す値を示します。'true' であればコントロールは現在リクエスタパネル上で可視状態にあります。

x（読取専用）

x にはリクエスタパネル上のコントロールの現在の X 座標値が入っています。

y（読取専用）

y にはリクエスタパネル上のコントロールの現在の Y 座標値が入っています。

w（読取専用）

w にはコントロールの幅が入っています。

h（読取専用）

h にはコントロールの高さが入っています。

メソッド

active([Boolean])

`active([Boolean])`メソッドは、コントロールのアクティブ状態を即座に設定します。オプション値`Boolean`にはコントロールがアクティブかどうかを指定します。値が省略されると、暗黙のうちに'`true`'であると仮定されます。

visible([Boolean])

`visible([Boolean])`メソッドは、コントロールの可視状態を即座に設定します。オプション値`Boolean`にはコントロールが可視状態かどうかを指定します。値が省略されると、暗黙のうちに'`true`'であると仮定されます。

position(column, row)

`position(column, row)`は、開かれているリクエスタパネル上のコントロールの位置を指定した列/行で設定します。再配置の動作は瞬時に行われます。このメソッド呼び出しの前後には、`visible()`メソッドを使用してコントロールの障害のない移動を確認してください。

例：

この例では、**Control Object Agents**を使用して簡単にインターフェイスを作成し、属性を取得する方法についてご紹介します。

```
generic
{
  // 全ての処理はここで行います
  reqbegin("<Requester Title>");
    c0 = ctlinteger("Integer Control",1);
    c1 = ctlnumber("Number Control",1.0);
    c2 = ctlstring("String Control","my string");
    c3 = ctlcheckbox("Checkbox Control",true);
    c4 = ctlpopup("Popup control",1,@"Item 1","Item 2","Item 3"@);
    return if !reqpost();
    info(c0.value);
    info(c1.x);
    info(c2.y);
    info(c3.w);
    info(c4.h);
  reqend();
}
```

第 17 章：SHADER OBJECT AGENTS

全てのレイアウトスクリプトと同様、**LS/PT**スクリプトの処理ポイントは`process()`関数です。`process()`関数はピクセルごとに呼び出されます。**LS/PT**ではスクリプトの`process()`関数に引数を一つ提供します。この引数は**Object Agent**のインスタンスであり、**Shader Object**として知られています。このオブジェクトには、アクセス可能な（場合によっては修正可能な）データメンバとメソッドが含まれます。

データメンバ

sx（読取専用）

`sx`は、最終画像におけるピクセル座標値のスポットのＸ座標値を表す数値です。(0,0）は左上隅の座標値となります。

sy（読取専用）

`sy`は、最終画像におけるピクセル座標値のスポットのＹ座標値を表す数値です。(0,0）は左上隅の座標値となります。

oPos[3]（読取専用）

`oPos[3]`は、オブジェクト座標におけるスポットの座標位置を表す数値です。

wPos[3]（読取専用）

`wPos[3]`は、ワールド座標におけるスポットの座標位置を表す数値です。

gNorm[3]（読取専用）

`gNorm[3]`は、ワールド座標におけるスポットのジオメトリ法線を表す値です。これはスポットにおける無修正のポリゴンの法線であり、スムージングやバンプマッピングなどによって修正される前の法線情報です。

spotSize（読取専用）

spotSize は、スポットの直径の近似値をあらわす数値です。エッジ上に見えるサーフェイス上のスポットは長く薄くなっているので、近似値となっています。テクスチャアンチエリアシングを計算する場合などに使用可能です。

raySource[3]（読取専用）

raySource[3]は、ワールド座標における視線レイが発生する起点を表す数値です。カメラになる場合が多いでしょうが、カメラである必要はありません。

rayLength（読取専用）

rayLength は、視線レイが空間を通ってこのスポットに到達するまでの距離を表す値です。

cosine（読取専用）

cosine は、スポットにおいて視線レイとサーフェイス法線との角度の余弦を表す値です。これはビューの跳ね返りを示し、スポットの近似サイズを測定します。

oXfrm[9]（読取専用）

oXfrm[9]は、オブジェクトからワールド座標変換行列を参照しています。他の方法でも計算可能ですが、ここでは速度用にメソッドであり、主として方向ベクトル用に使用するために用意されています。

wXfrm[9]（読取専用）

wXfrm[9]は、ワールドからオブジェクト座標変換行列を参照しています。他の方法でも計算可能ですが、ここでは速度用にメソッドであり、主として方向ベクトル用に使用するために用意されています。

objID（読取専用）

objID はシェーディングされるオブジェクトを表す **Object Agent** へのポインタです。

polNum（読取専用）

polNum は、シェーディングされるオブジェクトのポリゴン数を表す整数値です。これは通常のメッシュオブジェクトの場合にはポリゴン数を指しますが、非メッシュオブジェクトの場合には、他のサブオブジェクト情報を表現しています。

wNorm[3]

`wNorm[3]`は、ワールド座標におけるスポットの新規ジオメトリ法線です。この値を修正することで、ジオメトリを修正することなくサーフェイスをバンプ状に見せることが出来ます（バンプマッピング）。シェーダーは修正後にベクトルを正規化しておく必要があります。

color[3]

`color[3]`は、サーフェイスカラーを表す赤[1]、緑[2]、青[3]のパーセンテージを表す数値です。値1.0は100%に相当します。

luminous

`luminous`はサーフェイスの自己発光度を表す数値です（1.0は100%）。

diffuse

`diffuse`はサーフェイスの拡散レベルを表す数値です（1.0は100%）。

specular

`specular`はサーフェイスの反射光を表す数値です（1.0は100%）。

mirror

`mirror`はサーフェイスの鏡面反射率を表す数値です（1.0は100%）。

transparency

`transparency`はサーフェイスの透明度を表す数値です（1.0は100%）。

eta

`eta`はサーフェイスの屈折率を表す数値です（1.0は100%）。

roughness

`roughness`はサーフェイスの粗さを表す数値です（1.0は100%）。

メソッド

illuminate(light,position)

illuminate(light,position)関数は、カレントインスタンスにおいて、指定したライトから指定した位置に当たるライトのレイを表す、6個の数値の配列（色[1-3]と方向[4-6]）を返します。指定したワールド座標値にライトが全く照射されていない場合には、ゼロが返されます。色には（もし存在するならば）陰やフォールオフ、スポットライトコーン、ライト、それにポイントの間にある透明なオブジェクトの効果が含まれます。

raytrace(position,direction)

raytrace(position,direction)関数は、指定した位置から指定した方向に向かう（ワールド座標で）レイを追跡するために呼び出されます。関数はレイの長さを表す要素[1]と、その方向からくる色[2-4]を表す4つの要素を含む配列を返します。レイの長さが無限であれば-1.0を返します。使用されるdirectionは出て行くレイの方向であり、単位ベクトルに正規化されている必要があります。

raycast()

raycast()は本来、raytrace()と同じパラメータを受け取る速度向上版の関数ですが、レイの長さしか返しません。シェーディングは評価されることはなく、レイトレースの再帰も行われません。

例：

この例では**Shader Object**の使用法を解説しています。適切な例となるよう、このコードの大半はシェーダー用の計算を行っています。とくにprocess()関数内のShader Object saに注目してください。

```
// LS/PT: Blotch by Bob Hood
//
// Stick a colored spot on a surface
// (based on the sample Shader plugin
// shipped with the LightWave 4.0 SDK)
//
// Updated 04.22.98 Bob Hood
@version 2.0
color;
center;
radius;
softness;
r2, piOverR;
create
{
  color[1] = 0.9;
  color[2] = 0.0;
  color[3] = 0.2;
  center[1] = 0.0;
  center[2] = 0.0;
  center[3] = 0.0;
  radius = 1.5;
  softness = 0.5;
}
init
{
  r2 = radius * radius;
  piOverR = 3.1416 / radius;
}
flags
{
  return(COLOR);  // テクスチャのカラーの修正します
}
process: sa
// 'sa' は Shader Object のインスタンスです
{
  localr2 = 0.0;
  for(i = 1; i <= 3; i++)
  {
```

```
    d = sa.oPos[i] - center[i];
    d = d * d;
    if(d > r2)
      return;
    localr2 += d;
  }
  if(localr2 > r2)
    return;
  d = sqrt(localr2);
  d = cos(d * piOverR) * softness;
  if(d > 1.0)
    d = 1.0;
  a = 1.0 - d;
  for(i = 1; i <= 3; i++)
    sa.color[i] = sa.color[i] * a + color[i] * d;
}
options // ここがインターフェイスコードとなります
{
  reqbegin("Blotch");
    clr = <integer(color[1] * 255),
    integer(color[2] * 255),
    integer(color[3] * 255)>;
    cntr = <center[1],center[2],center[3]>;
    c1 = ctlrgb("Blotch color",clr);
    c2 = ctlvector("Blotch center",cntr);
    c3 = ctlnumber("Radius",radius);
    c4 = ctlnumber("Softness",softness);
    if(reqpost())
    {
      clr = getvalue(c1);
      color[1] = clr.x / 255;
      color[2] = clr.y / 255;
      color[3] = clr.z / 255;
      cntr = getvalue(c2);
      center[1] = cntr.x;
      center[2] = cntr.y;
      center[3] = cntr.z;
      radius = getvalue(c3);
      softness = getvalue(c4);
      r2 = radius * radius;
      piOverR = 3.1416 / radius;
    }
  reqend();
}
save: what, io
```

```
{
  if(what == OBJECTMODE)
  {
    io.writeNumber(color[1]);
    io.writeNumber(color[2]);
    io.writeNumber(color[3]);
    io.writeNumber(center[1]);
    io.writeNumber(center[2]);
    io.writeNumber(center[3]);
    io.writeNumber(radius);
    io.writeNumber(softness);
  }
}
load: what,io
{
  if(what == OBJECTMODE)   // オブジェクトファイルの処理
  {
    color[1] = io.readNumber();
    color[2] = io.readNumber();
    color[3] = io.readNumber();
    center[1] = io.readNumber();
    center[2] = io.readNumber();
    center[3] = io.readNumber();
    radius = io.readNumber();
    softness = io.readNumber();
  }
}
```

第 18 章 : SURFACE OBJECT AGENTS

Surface()コンストラクタは、既存のサーフェイス、またはオブジェクト用に新規作成したサーフェイス用に **Object Agent** を生成することが出来ます。コンストラクタは引数無しの呼び出しも可能で、その場合はシステム内で最初に定義されているサーフェイスが返されます。

```
surfObj = Surface();
```

Surface()コンストラクタが整数値付きで呼び出された場合には、システム内でインデックス値に対応するサーフェイスが返されます。

```
surfObj = Surface(1);
```

サーフェイス名称を引数として渡した場合、システムに既に定義されていれば、サーフェイスを返します。サーフェイスが存在しない場合、レイアウトでは'nil'が返されますが、モデラーではカレントオブジェクトに対し、その名称で新規サーフェイスを作成します。

```
surfName = Surface( "Surface Name" );
```

Mesh Object Agent を渡すと、コンストラクタはオブジェクトに定義されているサーフェイス名称のリストを返します。

```
meshObj = Mesh( "cow" );
surfNames = Surface(meshObj);
```

レイアウトでは、オブジェクトに対し新規サーフェイスを作成することが可能です。サーフェイスを受け取るためには、第1引数に **Mesh Object Agent** を、第2引数にサーフェイス名称を指定します。

```
meshObj = Mesh( "cow" );
surfNames = Surface(meshObj,  "New Surface" );
```

定数値

以下は **Surface Object Agent** で使用するため LScript で定義されているチャンネル定数値です。

SURFCOLR	Base Color	（ベクトル）
SURFLUMI	Luminosity	（数値）
SURFDIFF	Diffuse	（数値）
SURFSPEC	Specularity	（数値）
SURFREFL	Reflectivity	（数値）
SURFTRAN	Transparency	（数値）
SURFTRNL	Translucency	（数値）
SURFRIND	IOR	（数値）
SURFBUMP	Bump	（数値）
SURFGLOS	Glossiness	（数値）
SURFBUF1	Special Buffer 1	（数値）
SURFBUF2	Special Buffer 2	（数値）
SURFBUF3	Special Buffer 3	（数値）
SURFBUF4	Special Buffer 4	（数値）
SURFSHRP	Diffuse Sharpness	（数値）
SURFSMAN	Smoothing Angle	（度数）
SURFRSAN	Reflection Seam Angle	（度数）
SURFTSAN	Refraction Seam Angle	（度数）
SURFRBLR	Reflection Blur	（数値）
SURFTBLR	Refraction Blur	（数値）
SURFCLRF	Color Filter	（数値）
SURFCLRH	Color Highlights	（数値）
SURFADTR	Additive Transparency	（数値）
SURFAVAL	Alpha Value	（数値）
SURFGVAL	Glow Value	（数値）
SURFLCOL	Line Color	（ベクトル）
SURFLSIZ	Line Size	（数値）
SURFSIDE	Sidedness	（整数）
SURFGLOW	Glow	（ブール値）
SURFLINE	Render Outlines	（ブール値）
SURFRIMG	Reflection Image	（Image Object Agent）
SURFTIMG	Refraction Image	（Image Object Agent）
SURFVCOL	Vertex Coloring	（数値）

データメンバ

name

`name` はサーフェイス名称です。

メソッド

getValue(channel)

`getValue(channel)`は、指定したチャンネル（上記定義）に設定された値を返します。

getEnvelope(channel)

`getEnvelope(channel)`は、指定したチャンネルに割り当てられている **Envelope Object Agent** を返します。エンベロープが存在しない場合には'`nil`'を返します。

getTexture(channel)

`getTexture(channel)`は、指定したチャンネルに割り当てられている **Texture Object Agent** を返します。エンベロープが存在しない場合には'`nil`'を返します。

例：

この例では、**Surface Object Agent** の使用法を紹介しています。この例を適切に動作させるために、レイアウト上にLightWave のコンテンツから “ objects￥geography￥genearth￥earth.lwo ” を読み込んでください。

```
@version 2.5
@warnings
@script generic
objName = "Earth";
surfName = "GlobeSurface";
generic
{
  // Mesh Object Agent を作成
  meshObj = Mesh(objName);
  if(meshObj)
  {
    // Surface Object Agent を作成
    surfObj = Surface(surfName);
```

```
    if(surfObj)
    {
      // 値を取得
      colr = surfObj.getValue(SURFCOLR);
      lumi = surfObj.getValue(SURFLUMI);
      diff = surfObj.getValue(SURFDIFF);
      spec = surfObj.getValue(SURFSPEC);
      refl = surfObj.getValue(SURFREFL);
      trans = surfObj.getValue(SURFTRAN);
      // ここで値を表示
      info(colr);
    }
    else
        error("This is not the ", surfName, "surface!");
  }
  else
    error("This is not the ", objName, "object!");
}
```

第 19 章 : TEXTURE LAYER OBJECT AGENTS

TextureLayer Object Agentは、**Texture Object Agent**から生成され、Texture内部で定義されている各レイヤーに対する設定用プロキシとして提供されます。

定数値

以下は**TextureLayer Object Agent**で使用するためLScriptで定義されているチャンネル定数値です。

TXLRPOSITION	Position	（ベクトル）
TXLRROTATION	Rotation	（ベクトル）
TXLRSIZE	Size	（ベクトル）
TXLRFALLOFF	Falloff	（ベクトル）
TXLRPROJECT	Projection	（定数値）
TXLRAXIS	Texture Axis	（整数）
TXLRWWRAP	Width Wrap	（数値）
TXLRHWRAP	Height Wrap	（数値）
TXLRCOORD	Coordinate System	（整数: 1== オブジェクト座標, 2== ワールド座標）
TXLRIMAGE	Image	（Image Object Agent）
TXLRVMAP	VMap	（VMap Object Agent）
TXLREFOBJ	Reference Object	（Layout Object Agent）
TXLROPACITY	Opacity	（数値）
TXLRANTIALIAS	Antialias	（ブール値）
TXLRAAVALUE	Antialias Threshold	（数値）
TXLRPIXBLEND	Pixel Blending	（ブール値）
TXLRWREPEAT	Width Repeat	（定数値）
TXLRHREPEAT	Height Repeat	（定数値）

テクスチャの TXLRPROJECT 設定用に定義されている定数値は

TXPJPLANAR
TXPJCYLINDRICAL
TXPJSPHERICAL
TXPJCUBIC
TXPJFRONT
TXPJUVMAP

テクスチャの TXLRWREPEAT と TXLRHREPEAT 設定用に定義されている定数値は

TXRPRESET
TXRPREPEAT
TXRPMIRROR
TXRPEDGE

データメンバ

以下のデータメンバが **TextureLayer Object Agent** によりエクスポートされます。

type

type には、レイヤーの種類が入っています
(TXTRIMAGE, TXTRPROCEDURE もしくは TXTRGRADIENT)。

メソッド

以下のメソッドが利用可能です。

getValue(channel)

getValue(channel)は、レイヤーにおいて指定したチャンネルの値を取得します。

setValue(channel, value)

setValue(channel, value)は、指定したレイヤーチャンネルに対し値を設定します。

例：

この例では、**TextureLayer Object Agents**を使用してテクスチャ内で使用されている画像のファイル名称を表示します。適切に動作するためにレイアウト上へオブジェクト“ objects￥geography￥genearth￥earth.lwo ”を読み込んで実行してください。

```
@version 2.5
@warnings
@script generic
objName = "Earth";
surfName = "GlobeSurface";
generic
{
  // Mesh Object Agent を作成
  meshObj = Mesh(objName);
  if(meshObj)
  {
    // Surface Object Agent を作成
    surfObj = Surface(surfName);
    if(surfObj)
    {
      // Texture Object Agent を作成
      texObj = surfObj.getTexture(SURFCOLR);
      if(texObj)
      {
        // TextureLayer Object を作成
        texLayerObj = texObj.firstLayer();
        if(texLayerObj.type == TXTRIMAGE)
        {
          imageObj = texLayerObj.getValue(TXLRIMAGE);
          info(imageObj.filename(1));
        }
        else
          error("Not a valid image layer!");
      }
      else
        error("Not a valid texture layer!");
    }
    else
      error("This is not the ", surfName, " surface!");
  }
  else
    error("This is not the ", objName, " object!");
}
```

第 20 章：TEXTURE OBJECT AGENTS

Texture Object Agentは、**Surface Object Agent**から（`getTexture()`メソッドを使用して）生成されます。このObject Agentは、サーフェイスで特定のチャンネルに割り当てられているテクスチャのインターフェイスを提供します。テクスチャには単数もしくは複数のレイヤーが含まれ、それぞれが独自の設定を持っています。このレイヤーを積みかさねることで、サーフェイスの外観を表現していきます。

定数

以下は**Texture Object Agent**で使用するためLScriptで定義されているチャンネル定数値です。

`TXTRIMAGE`	Image Texture type（画像）
`TXTRPROCEDURE`	Procedural Texture type （プロシージャル）
`TXTRGRADIENT`	Gradient Texture type （グラディエント）

データメンバ

現在Texture Object Agentsからエクスポートされるデータメンバはありません。

メソッド

setChannelGroup(Channel Group Object Agent)

`setChannelGroup（Channel Group Object Agent）`は、指定されたObject Agentを使用してテクスチャ用のチャンネルグループを設定します。テクスチャレイヤーにおけるパラメータ用に作成されたエンベロープは、このグループに属しています。

getChannelGroup()

`getChannelGroup()`は、テクスチャに対するChannel Group Object Agentを返します。

firstLayer()

`firstLayer()`は、テクスチャ内で最初に定義されているTextureLayer Object Agentを返します。

nextLayer(TextureLayer Object Agent)

`nextLayer(TextureLayer Object Agent)`は、テクスチャに対する指定したTextureLayerの次にくるTextureLayer Object Agentを返します。

addLayer(layer type)

`addLayer(layer type)`は、指定したタイプ（`TXTRIMAGE`、`TXTRPROCEDURE`、または`TXTRGRADIENT`）で新規レイヤーを作成し、そのTextureLayer Object Agentを返します。

例：

この例では、**Texture Object Agent**を使用して、既存のチャンネルに対し新規テクスチャを追加しています。この例を適切に動作させるため、LightWaveのコンテンツからオブジェクトファイル“objects¥geography¥genearth¥earth.lwo”をレイアウト上に読み込んでください。

```
@version 2.5
@warnings
@script generic
objName = "Earth";
surfName = "GlobeSurface";
generic
{
  // Mesh Object Agent の作成
  meshObj = Mesh(objName);
  if(meshObj)
  {
    // Surface Object Agent の作成
    surfObj = Surface(surfName);
    if(surfObj)
    {
      // Texture Object Agent の作成
      texObj = surfObj.getTexture(SURFCOLR);
      if(texObj)
      {
        // 新規 Texture layer の追加
        texObj.addLayer(TXTRIMAGE);
      }
      else
        error("Not a valid texture layer!");
    }
    else
      error("This is not the ", surfName, " surface!");
  }
  else
    error("This is not the ", objName, " object!");
}
```

第 21 章 : IMAGE FILTER OBJECT AGENTS

Image Filter LScript のアーキテクチャは、画像のピクセルデータにアクセスし修正するためのメカニズムを提供します。**ImageFilter Object Agent** は、引数が一つだけ定義されている場合には Image Filter スクリプトの `process()` 関数へと渡されます。

```
process: ifo
{
...
}
```

データメンバ

以下のデータメンバが **ImageFilter Object Agent** からエクスポートされます。

width（読取専用）

`width` はカレントバッファの幅（ピクセル値）です。

height（読取専用）

`height` はカレントバッファの高さ（ピクセル値）です。

frame（読取専用）

`frame` は画像に対するカレントフレーム番号です。

start（読取専用）

`start` は画像に対するレンダリング開始時刻です。

end（読取専用）

`start` は画像に対するレンダリング終了時刻です。

red[]

`red[]`は、イメージに対する赤のピクセルバッファを浮動小数点数形式で表します。インデックスは[列、行]形式となります。

green[]

`green[]`は、イメージに対する緑のピクセルバッファを浮動小数点数形式で表します。インデックスは[列、行]形式となります。

blue[]

`blue[]`は、イメージに対する青のピクセルバッファを浮動小数点数形式で表します。インデックスは[列、行]形式となります。

alpha[]

`alpha[]`は、イメージに対するアルファのピクセルバッファを浮動小数点数形式で表します。インデックスは[列、行]形式となります。

special[]（読取専用）

`special[]`はスペシャルバッファの値をあらわします。インデックスは[列、行]形式となります。

luminous[]（読取専用）

`luminous[]`は自己発光度バッファの値をあらわします。インデックスは[列、行]形式となります。

diffuse[]（読取専用）

`diffuse[]`は拡散レベルバッファの値をあらわします。インデックスは[列、行]形式となります。

specular[]（読取専用）

`specular[]`は反射光バッファの値をあらわします。インデックスは[列、行]形式となります。

mirror[]（読取専用）

`mirror[]`は鏡面反射率バッファの値をあらわします。インデックスは[列、行]形式となります。

trans[]（読取専用）

`trans[]`は透明度バッファの値をあらわします。インデックスは[列、行]形式となります。

rawred[]（読取専用）

`rawred[]`は未加工の赤のバッファの値をあらわします。インデックスは[列、行]形式となります。

rawgreen[]（読取専用）

`rawgreen[]`は未加工の緑のバッファの値をあらわします。インデックスは[列、行]形式となります。

rawblue[]（読取専用）

`rawblue[]`は未加工の青のバッファの値をあらわします。インデックスは[列、行]形式となります。

shading[]（読取専用）

`shading[]`はシェーディングバッファの値をあらわします。インデックスは[列、行]形式となります。

shadow[]（読取専用）

`shadow[]`はシャドーバッファの値をあらわします。インデックスは[列、行]形式となります。

geometry[]（読取専用）

`geometry[]`はジオメトリバッファの値をあらわします。インデックスは[列、行]形式となります。

depth[]（読取専用）

`depth[]`はデプスバッファの値をあらわします。インデックスは[列、行]形式となります。

diffshade[]（読取専用）

`diffshade[]`はシェーディング付き拡散レベルバッファの値をあらわします。インデックスは[列、行]形式となります。

specshade[]（読取専用）

`specshade[]`はシェーディング付き鏡面反射率バッファの値をあらわします。インデックスは[列、行]形式となります。

motionx[]（読取専用）

`motinox[]`はMotionXのバッファの値をあらわします。インデックスは[列、行]形式となります。

motiony[]（読取専用）

`motinoy[]`はMotionYのバッファの値をあらわします。インデックスは[列、行]形式となります。

reflectred[]（読取専用）

`reflectred[]`は反射の赤バッファの値をあらわします。インデックスは[列、行]形式となります。

reflectgreen[]（読取専用）

`reflectgreen[]`は反射の緑バッファの値をあらわします。インデックスは[列、行]形式となります。

reflectblue[]（読取専用）

`reflectblue[]`は反射の青バッファの値をあらわします。インデックスは[列、行]形式となります。

メソッド

以下のメソッドが利用可能です。

copy(left, top, right, bottom)

`copy(left, top, right, bottom)`は、カレントバッファにおいて特定領域内部（内包）にあるイメージデータを使用して新規イメージバッファを作成します。返されるバッファIDは`select()`メソッドなどで使用します。

paste(bufferid, left, top)

`paste(bufferid, left, top)`は、指定したバッファ内にあるイメージデータを、現在選択されているバッファ内の指定した左上オフセット値の箇所に貼り付けます。バッファIDは前回`copy()`メソッドの呼び出しで返された値でなくてはなりません。

select ([bufferid])

`select ([bufferid])`は、カレントの作業バッファとしてイメージバッファを選択します。データメンバとメソッドは全てこの呼び出しの後に、選択バッファに対し処理を行います。何の引数も渡さない場合には、メインイメージバッファがカレントバッファとなります。

例１：

ImageFilter Object Agentを使用すると、LightWaveに提供されているImage Filter LScriptである"Black & White"の`process()`関数は、以下のように記述することが出来ます。

```
process: ifo
{
  for(i = 1;i <= ifo.height;++i)
  {
    for(j = 1;j <= ifo.width;++j)
    {
      average = (ifo.red[j,i] + ifo.green[j,i] + ifo.blue[j,i]) / 3;
      ifo.red[j,i] = average;
      ifo.green[j,i] = average;
      ifo.blue[j,i] = average;
    }
  }
}
```

例２：

以下の例では`copy()`、`paste()`それに`select()`メソッドを使用して、イメージの縦半分を入れ替えています。

```
process: ifo
{
  buf1 = ifo.copy(1,1,ifo.width / 2,ifo.height);
  buf2 = ifo.copy(ifo.width / 2,1,ifo.width,ifo.height);
  ifo.select(buf1);
  buf_width = ifo.width;
  ifo.select();
  ifo.paste(buf2,1,1);
  ifo.paste(buf1,buf_width,1);
}
```

第 22 章： DISPLACEMENT MAP OBJECT AGENTS

データメンバ

oPos[3]（読取専用）

oPos[3]は、変位処理前のポイントの位置を表す数値です。これらの数値には修正が加えられておらず、進行時間に基づきオフセット値を計算するために使用されるものです。

source[3]

source[3]は、カレントフレーム/タイムにおいて処理されるポイントの新しい座標値を表す値です。

メソッド

Displacement Object Agentから提供されるメソッドはありません。

例：

例“ LazyPoints.ls ”では、Displacement Map クラスのスクリプトを使用して **Displacement Map Object Agent**の使用法を紹介しています。

```
//-----------------------------------------------
--
// LazyPoints LS by Bob Hood
//
// (based on LazyPoints by Allen Hastings)
//
@version 2.3
// 大域変数の設定
curFrame,curTime,curId;
lagRate = .25;
newtime: id, frame, time
{
```

```
    // 大域変数の更新
    curFrame = frame;
    curTime = time;
    if(!curId) curId = id;
  }
  process: da
  {
    // da Object Agent を使用して ポイントの元々の位置
    // (-ピボットポイント位置)を取得します
    original = <da.oPos[1],da.oPos[2],da.oPos[3]> - curId.getPivot(curTime);
    // lagtime を計算します
    lagTime = curTime - lagRate * vmag(original);
    // モーション情報を集めます
    xax = curId.getRight(lagTime);
    yax = curId.getUp(lagTime);
    zax = curId.getForward(lagTime);
    pos = curId.getWorldPosition(lagTime);
    // da Object Agent を使用して
    // ポイントの新しい座標値を設定します
    da.source[1] = pos.x + original.x * xax.x + original.y * yax.x +
original.z * zax.x;
    da.source[2] = pos.y + original.x * xax.y + original.y * yax.y +
original.z * zax.y;
    da.source[3] = pos.z + original.x * xax.z + original.y * yax.z +
original.z * zax.z;
  }
  options
  {
    // 簡単なインターフェイスを作成します
    reqbegin("LazyPoints LS");
      c1 = ctlnumber("Lag Rate (sec/meter)",lagRate);
      return if !reqpost();
      lagRate = c1.value;
    reqend();
  }
```

第 23 章：
OBJECT REPLACEMENT OBJECT AGENTS

LScript Object Replacement（LS/OR）スクリプトを使用すると、レイアウト上でフレームごとにオブジェクトを自由に置き換えることが可能になります。

LS/OR プラグインの`process()`は単一引数を受け取ります。この引数は**Replacement Object Agent**のインスタンスです。**LS/OR Replacement Object Agent**は、以下のデータメンバを提供します。

データメンバ

objID（読取専用）

`objID`は、置き換えるジオメトリを持つオブジェクトの Object Agent です。

curFrame（読取専用）

`curFrame`は、現在読み込まれているジオメトリに対するフレーム番号を表す整数値です。

curTime（読取専用）

`curFrame`は、現在読み込まれているジオメトリに対するタイムインデックスを表す整数値です。

newFrame（読取専用）

`newFrame`は、次のステップのフレーム番号を表す整数値です。この新しいフレーム番号に対し異なるオブジェクトを検索しに行く必要がある場合には、新規ジオメトリが読み込まれることになります。

newTime（読取専用）

`newTime`は、次のステップのタイムインデックスを表す整数値です。この新しいフレーム番号に対し異なるオブジェクトを検索しに行く必要がある場合には、新規ジオメトリが読み込まれることになります。ネットワークレンダリングではレンダラーが連続していない時刻間で飛んでしまう可能性があるため、`'curTime'`と`'newTime'`はシーケンシャルではありません。

curType（読取専用）

curType は、レンダリングが行われるカレントタイプを示す定数値です。スクリプトはこの値を調べることにより、インタラクティブなプレビューと実際のレンダリングに対し異なるジオメトリを供給することが出来ます。この値には NONE,PREVIEW もしくは RENDER を指定します。現在時刻に対しジオメトリが何も読み込まれていない場合には、NONE が現れます。PREVIEW はレイアウトプレビューが生成されていることを示し、RENDER は完全なレンダリングが行われるときに使用されます。

newType（読取専用）

newType は、次のフレーム / タイムに処理されるレンダリングの種類を示す定数値です。このメンバは NONE,PREVIEW もしくは RENDER となります。

curFilename（読取専用）

curFilename は、現在読み込まれているオブジェクトのジオメトリファイルを表す文字列であり、ジオメトリが何も読み込まれていない場合には'nil'が返されます。

newFilename

newFilename は、新しい時刻に、このアイテムに対しジオメトリとして読み込まれるべき新しいオブジェクトファイルの、ファイル名称を表す文字列です。これはスクリプトにより設定される唯一のデータメンバです。新規ジオメトリが現在読み込まれているものと異なる場合にのみ設定されます。新規ジオメトリの読み込みは多大なオーバーヘッドを引き起こすためです。

第 24 章：PARTICLE OBJECT AGENTS

Particle Object Agentは、プラグインのAPI Particle システムサービスの基幹をなすインターフェイスを提供します。**Particle Object Agent**インスタンスは、コンストラクタ`Particle()`を使用し作成されます。このコンストラクタを使用して、Particleデータベースの新規インスタンスを作成したり、提供される引数によっては既存のParticleシステムを要求することが出来ます。

有効な**Layout Object Agent**がコンストラクタに提供されていると、オブジェクトに対して有効なParticleシステムのリストが返されます。オブジェクトに対し有効になっているParticleシステムがない場合には'`nil`'が返されます。

```
obj = Mesh("Cow");
part = Particle(obj) || error("No particle systems active!");
```

新規Particleシステムを作成する際は、パーティクルタイプを指定し、各パーティクルを作成し保持するためのバッファのリストを指定する必要があります。Particleタイプには`PART_PARTICLE`（単一パーティクル）と`PART_TRAIL`（尾っぽ付きパーティクル）があります。LScriptでの大半の定数値と同様、これらもまた、等価となる文字列"`particle`"と"`trail`"でも指定可能です。

以下のバッファは各パーティクルに対し割り当て可能です。

```
PART_POSITION     (もしくは"position")
PART_SIZE         (もしくは"size")
PART_SCALE        (もしくは"scale")
PART_ROTATION     (もしくは"rotation")
PART_VELOCITY     (もしくは"velocity")
PART_AGE          (もしくは"age")
PART_FORCE        (もしくは"force")
PART_PRESSURE     (もしくは"pressure")
PART_TEMPERATURE  (もしくは"temperature")
PART_MASS         (もしくは"mass")
PART_LINK         (もしくは"link")
PART_ID           (もしくは"id")
PART_ENABLE       (もしくは"enable")
PART_RGBA         (もしくは"rgba")
PART_COLLISION    (もしくは"collision")
```

パーティクルは三つの状態、`PART_ALIVE`、`PART_DEAD`または`APRT_LIMBO`で存在します。これらの値は`PART_ENABLE`バッファの読み込み/書き込み時に使用されます。

データメンバ

count

`count` は、インスタンス内で定義される現在のパーティクル数（整数値）を返します。

メソッド

save()

`save()`は、保存状態中にパーティクル特有のデータを書き込みます。このメソッドは事前に定義されている `save()`スクリプト関数内でのみ使用が可能です。

load()

`load()`は、読み込み状態中にパーティクルデータを読み込みます。このメソッドは事前に定義されている `load()`スクリプト関数内でのみ使用が可能です。

attach(Object Agent)

`attach(Object Agent)`は、**Layout Object Agent**へパーティクルインスタンスを割り当て、HyperVoxelsのようなシステムもパーティクルデータにアクセスできるようになります。

detach(Object Agent)

`detach(Object Agent)`は、オブジェクトからパーティクルインスタンスの割り当てを解除します。

reset()

`reset()`は、インスタンスから全てのパーティクルを削除します。

addParticle()

`addParticle()`は、インスタンス内に新規パーティクルを生成します。新規パーティクルのインデックス値が返されます。

remParticle(index)

`remParticle(index)`は、インスタンスから指定されたインデックス値を持つパーティクルを除去します。

setParticle(index,bufid,value)

`setParticle(index,bufid,value)`は、指定したパーティクルインデックス用の指定したバッファにデータを設定します。

getParticle(index,bufid)

`getParticle(index,bufid)`は、指定したパーティクルインデックス用のバッファ内にあるデータを取得します。

例：

ここに、パーティクルSDKの例と同じ動作を行うLScriptを紹介します。オブジェクトに対してこのスクリプトを適用し、HyperVoxelでオブジェクトを有効にしてみると確認できます。

```
@warnings
@version 2.3
@script displace
part;
create: id
{
  part = Particle("particle",@"position","size","enable","rgba"@);
  part.attach(id);
}
newtime: id, frame, time
{
  part.reset();
}
flags { return(WORLD); }
process: da
{
  i = part.addParticle();
  part.setParticle(i,"position",<da.oPos[1],da.oPos[2],da.oPos[3]>);
  part.setParticle(i,"rgba",100,150,200,255);
  part.setParticle(i,"enable",PART_ALIVE);
}
```

第 25 章：MOTION OBJECT AGENTS

Item Animation LScript は、`process()` 関数に対し引数を三つ提供します。第 1 引数は **Motion Object** のインスタンスです。第 2、第 3 パラメータは現在の呼び出しにおけるフレーム番号（整数）とタイムインデックス（数値）です。

データメンバ

objID（読取専用）

`objID` は、このプラグインが割り当てられる **Object Agent** のポインタです。

メソッド

get(attribute,time)

`get(attribute,time)` 関数は、指定した'`time`'における'`attribute`'の値を表すベクトルです。'`attribute`'には `POSITION` または `WPOSITION`、`ROTATIONSCALING` が指定可能です。'`time`'は、値を取得したい時刻におけるタイムインデックスです。`ROTATION` の場合、値は度数となります。`SCALING` の値は乗数となりますので、1.0 だと何の処理も行われていないことを示します。

set(attribute,value)

`set(attribute,value)` 関数を使うと、特定の'`attirbute`'（`POSITOION`、`ROTATION` または `SCALING`）に対し、指定したベクトルの値'`value`'を設定することが出来ます。

例：

このスクリプトは、他のオブジェクトへターゲット設定されているボーンに対して適用します。スクリプトはボーンに均一な拡大縮小を使用することで、ターゲットを動かしてもボーンがくっついていくようにしてあります。

```
//-----------------------------------------------
// Bone Target by Bob Hood
//
@version 2.3
myObj;
create: obj
{
  // 大域変数 myObj に Object Agent を保存します
  myObj= obj;
  setdesc("Bone Target - By Bob Hood");
}
flags
{
  return(IA_AFTERIK);
}
process: ma, frame, time
{
  // ゴールがあるかどうかを確かめます
  if(myObj.goal == nil)
  return;
  dist = vmag(myObj.goal.param(WPOSITION,time)

  ma.get(WPOSITION,time));
  // ボーンが縮小しないようにします
  if((scale = (dist / myObj.restlength)) < 1.0)
    scale = 1.0;
  // ma Object Agent を使用して SCALING 属性を設定します
  ma.set(SCALING,<scale,scale,scale>);
}
```

第 26 章：CHANNEL OBJECT AGENTS

Channel Object Agentは、それぞれのチャンネル機能に対し構造化されたインターフェイスを提供します。`firstChannel()`と`nextChannel()`メソッドは、**Channel Object**タイプを返します。場合によって`Channel Object Agent`にはUDFが提供されることもあります（Channel Filter LScript用の`create()` UDF）。

スクリプトは直接キーフレームIDを使用することはなく、整数データタイプとして返します。キーフレームIDはIDを要求するメソッドに対する引数としてのみ提供されます。

データメンバ

name

`name`は、レイアウトインターフェイスに表示されるチャンネル名称です（例：**Graph Editor**）。

type

`type`はチャンネルの種類です。`NUMBER`、`DISTANCE`、`PERCENT`もしくは`ANGLE`となります。

keyCount

`keyCount`は、チャンネルに割り当てられているエンベロープ内のキー総数を返します。

keys[]

`keys[]`は、エンベロープ内の全てのキーのキーフレームIDが入っている線形配列を返します。要素数は常に'`keuCount`'で返される値となります。

preBehavior

`preBehavior`には、エンベロープにつけられている前の振る舞いの値が入っています。この値は`CHAN_RESET`、`CHAN_CONSTANT`、`CHAN_REPEAT`、`CHAN_OSCILLATE`、`CHAN_OFFSET`もしくは`CHAN_LINEAR`となります。文字列‘`reset`,’、‘`constant`,’、‘`repeat`,’、‘`oscillate`,’‘`offset`,’、もしくは‘`linear`’でも可能です。

大半のデータメンバとは異なり、このデータメンバは読取専用ではありません。キーに対する上記定数に値を割り当てることで実際にエンベロープの前の振る舞いを修正します。

postBehavior

`postBehavior` には、エンベロープに割り当てられている後の振る舞いが入っています。名称を覗けば、このデータメンバは'`preBehavior`'データメンバと全く同じ働きをします。値を返したり、エンベロープに対して設定されている後の振る舞いの値を修正したりします。

メソッド

value(time)

`value(time)`メソッドを使うと、指定した時刻におけるチャンネルを評価することが出来ます。戻り値はチャンネルの種類に基づいて変換処理された浮動小数点数値となります。

event(UDF)

`event(UDF)`メソッドは、チャンネル上にイベントが発生したときにいつでも呼び出されるコールバック関数を起動します。イベントにはエンベロープにおけるキーフレームの作成や削除、キーフレーム値の修正などがあります。

keyExists(<time>)

`keyExists(<time>)`は、指定した時刻においてキーが存在しなければ'`nil`'を、存在する場合にはキーフレーム ID を返します。

setKeyValue(<key>,<value>)

`setKeyValue(<key>,<value>)`は、指定したキーフレームに対し指定した値を設定します。`<value>` は常に浮動小数点数値となります。

setKeyCurve(<key>,<shape>)

`setKeyCurve(<key>,<shape>)`には、キーフレームを評価するときの手前の曲線の種類を設定します。これには定数値 `CHAN_TCB`、`CHAN_HERMITE`、`CHAN_BEZIER`、`CHAN_LINEAR`、`CHAN_STEPPED` を、または文字列 ' `TCB`, ' ' `Hermite`, ' ' `Bezier`, ' ' `Linear`, ' もしくは ' `Stepped` ' を指定することが出来ます。

setKeyHermite(<key>,<parm1>,<parm2>,<parm3>,<parm4>)

`setKeyHermite(<key>,<parm1>,<parm2>,<parm3>,<parm4>)`を使用すると、Hermite 曲線の 4 個の係数を設定することが出来ます。使用法がわからない場合には、このメソッドを使う必要はありません。

setKeyTension(<key>,<value>)

`setKeyTension(<key>,<value>)`は、指定したキーフレームにおける張力を設定します。

setKeyContinuity(<key>,<value>)

`setKeyContinuity(<key>,<value>)`は、指定したキーフレームにおける連続性を設定します。

setKeyBias(<key>,<value>)

`setKeyBias(<key>,<value>)`は、指定したキーフレームにおける傾斜を設定します。

getKeyValue(<key>)

`getKeyValue(<key>)`は、指定したキーフレームに割り当てられている浮動小数点数値を返します。

getKeyTime(<key>)

`getKeyTime(<key>)`は指定したキーフレームに対するタイムインデックスを返します。

getKeyCurve(<key>)

`getKeyCurve(<key>)`は指定したキーフレームに対する手前の曲線のタイプを返します。返される定数値のリストについては、`setKeyCurve()`のエントリを参照してください。

getKeyHermite(<key>)

`getKeyHermite(<key>)`は、指定したキーフレームで現在使用されている４個のHermite曲線の係数を返します。`KeyCurve()`から返される値が`CHAN_HERMITE`ではない場合には、このメソッドを呼び出す必要はないでしょう。

getKeyTension(<key>)

`getKeyTension(<key>)`は、キーフレームに対する現在の張力の設定を返します。

getKeyContinuity(<key>)

`getKeyContinuity(<key>)`は、キーフレームに対する現在の連続性の設定を返します。

getKeyBias(<key>)

`getKeyBias(<key>)`は、キーフレームに対する現在の傾斜の設定を返します。

createKey(<time>,<value>)

`createKey(<time>,<value>)`メソッドは、指定したタイムインデックスの箇所に指定した初期値で新規キーフレームを作成します。キー作成に成功したらキーIDが返されますが、失敗した場合には'`nil`'が返されます。このメソッドを使用すると、'`keyCount`'と'`kes[]`'データメンバは即時に更新されます。

deleteKey(<key>)

キーフレームは、`deleteKey(<key>)`メソッドでチャンネルのエンベロープからキーフレームを削除することが出来ます。また、このメソッドを使用すると'`keyCount`'と'`keys[]`'データメンバもリアルタイムに更新されますので、これらのデータメンバに基づいてループ内で使用している場合には、非常に気をつける必要があります。

例：

以下のChannel Filterスクリプトでは**Channel Object Agent**の使用法を紹介しています。

```
@warnings
@script channel
@version 2.0
create: channel
{
  light = Light();
  c = light.firstChannel();
  while(c)
  {
    last if c.name == "Position.X";
    c = light.nextChannel();
  }
  // このチャンネルにおいて修正と同期を保っています
  c.event("lightEvent");
}
process: ca, frame, time
{
  ca.set(0.0);
}
lightEvent: channel,
// Channel Object Agent イベント
// イベントコード
{
  // Light の "Position.X" チャンネルに発生しました
  info(event);
}
```

第 27 章：CHANNEL GROUP OBJECT AGENTS

Channel Group Object Agentsは、定義されているチャンネルグループへのアクセスを可能にします。コンストラクタ`ChannelGroup()`は**Channel Group Object Agent**を返します。この**Object Agent**は、インターフェイスという点においては標準のLayout Object Agentと似通っていますが、限られたメソッドとデータメンバのサブセットのみをサポートしています。この主な目的はチャンネルグループを列挙する手段を提供することです。チャンネルグループには"隠れている"ものもあります（例えば実際のオブジェクトに割り当てられていないチャンネルなど）。

`ChannelGroup()`はレイアウトにチャンネルグループが存在しない場合（ありそうにないような状況ですが、良質のプログラムに興味があるのであれば、この可能性も考慮に入れるべきです）、`ChannelGroup()`は'`nil`'を返します。

チャンネルグループの名称を指定せずに`ChannelGroup()`を呼び出すと、システムで最初に定義されているチャンネルグループが返されます。これを起点とし、**Layout Object Agent**と同様に、既存の全チャンネルグループを`next()`メソッドを使用し、反復していくことが可能です。チャンネルグループが存在しない箇所まできたら'`nil`'が返されます。

また`ChannnelGroup()`コンストラクタは指定したチャンネルグループに対応する**Object Agent**を生成するため、チャンネルグループID（文字列）も受け付けています。例えば**LW_Master Channel**プラグインで生成されたチャンネルグループにアクセスするためには、以下のようなコードを使用します.

```
group = ChannelGroup("MC");
```

もちろん、**LW_MasterChannel**プラグインが起動されていない場合には、'`nil`'が返されます。

サブグループ処理に取り組む場合、`ChannelGroup()`は既存の**Channel Group Object Agent**を受け取ります。Agentが提供されているときには、コンストラクタはそれを親として更なるChannel Groupの場所を捜し求め、親の直下に定義されている一番目のサブグループを返します。このアクションを使用し定義されている全てのChannelGroupにアクセスするには、再帰処理が必要となります（例参照）。

データメンバ

name

`name` は、Object Agent がプロキシとして提供しているチャンネルグループの名称です。

parent

`parent` には、カレントの **Object Agent** の親に当たるチャンネルグループが入っています。親が存在しないようであれば、このデータメンバは'`nil`'となります。

メソッド

firstChannel(), nextChannel()

`firstChannel()` と `nextChannel()` は、それぞれ **Channel Object Agent** を反復していきます（**Channel Obejct Agent** で利用可能なメソッドとデータメンバを参照）。

next()

`next()` は、リスト内における次のチャンネルグループを返します。

例：

```
@warnings
@script channel
@version 2.4.1
ChGroup;
create: channel
{
  ChGroup = nil;
  // 再帰的にチャンネルグループを探しに行きます
  GetGroupName(channel.id,ChannelGroup());
  // チャンネルグループは見つかりましたか？
  if(ChGroup != nil)
  info(ChGroup.name + "." + channel.name);
}
GetGroupName: cid, chgroup
{
  if(ChGroup) return;
  while(chgroup)
  {
```

```
    chchannel=chgroup.firstChannel();
    while(chchannel)
    {
      if (chchannel.id == cid)
      {
        ChGroup=chgroup;
        last;
      }
      chchannel=chgroup.nextChannel();
    }
    last if ChGroup;
    // 再帰呼び出し
    GetGroupName(cid,ChannelGroup(chgroup));
    chgroup=chgroup.next();
  }
}
```

第 28 章：ENVELOPE OBJECT AGENTS

Envelope Object Agents は Envelope() コンストラクタを使用して作成されます。Envleope() コンストラクタは三つの引数を受け取りますが、最後の一つはオプションとなります。

```
Envelope(<string>,<type>[,<group>])
```

<string> は新規エンベロープの対する ID です。

<type> は CHAN_NUMBER、CHAN_DISTANCE、CHAN_PERCENT、もしくは CHAN_ANGLE です。

<group> はオプション引数です。新規エンベロープが存在するグループを識別するための文字列です。指定した <group> が存在しない場合は、LScript でそのグループを作成します。

データメンバ

Envelope Object Agent は、**Channel Object Agent** に付加されているものと同じメソッドとデータメンバを使用します（前章参照）。それに加え、以下のエンベロープ特有のメソッドも提供しています。

メソッド

copy(<Envelope>)

copy(<Envelope>) メソッドは、指定された **Envelope Object Agent** にあるインスタンス値へコピーします。

edit()

edit() は、ユーザーとインタラクティブにやり取りするためにエンベロープのコンテンツと設定を表示します。

save(), load()

`save()`と`load()`メソッドは、エンベロープの設定とデータをシーンファイルに保存/読み込みを行います。これらは文脈依存なメソッド、つまりレイアウトの特定の局面においてのみ呼び出されるということです（`save()`メソッドは`save()` UDFから、`load()`メソッドは`load()` UDFからのみ呼び出し可能です）。それぞれのコンテキストの範囲外でメソッドを使用しようとすると、とんでもない目にあってしまいますよ。

persist([<Boolean>])

`persist([<Boolean>])`メソッドを使うと、エンベロープ（とグループ）をレイアウトの内部チャンネルグループリストに残すことが可能です。そうしないとLScriptは、スクリプトが終了した段階でスクリプトにより生成されたエンベロープ（とグループ）を自動的に破棄します。このメソッドが引数無しで呼び出されると、エンベロープは存続するというフラグが立てられます。ブール値'`false`'は、スクリプトが終了した段階でLScriptによりエンベロープの除去を許可します。

第 29 章：CUSTOM OBJECT AGENTS

Animation と同様、**Custom Object** スクリプトもまた、`create()` UDF のオプション引数を受け取ることが可能です。オプション引数はスクリプトが割り当てられるオブジェクトの **Object Agent** です。**Custom Object** スクリプトもまた、`init()`、`newtime()`それに`cleanup()` UDF をサポートしています。

Custom Object の `process()` UDF は単一引数を受け取ります。引数は、基幹となる Custom Object 機能のインターフェイスを提供する Custom Object Agent です。以下のデータとメソッドをエクスポートします。

データメンバ

view（読取専用）

`view` は読取専用のデータメンバであり、どのビューの描画に影響を与えるのかを指定します。値は以下の通りです。

`VIEW_ZY`
`VIEW_XZ`
`VIEW_XY`
`VIEW_PERSP`
`VIEW_LIGHT`
`VIEW_CAMERA`
`VIEW_SCHEMA`

flags[]（読取専用）

`flags[]`は、カレントの Custom インスタンスに対する設定が入っている読取専用の配列です。現在は要素は 1 個のみ（flags[1]）で、カスタムオブジェクトが選択状態で描画されるのならば '`true`'、それ以外では'`false`'とします。

setColor(<r,g,b>|r,g,b[,a])

`setColor(<r,g,b>|r,g,b[,a])`は、このコマンド以降で使用する描画アクションの色を設定します。

setPattern(pat)

`setPattern(pat)`は、描画パターン（デフォルトでは`PATTERN_SOLID`）を設定します。指定可能な値は以下のとおりです。

```
PATTERN_SOLID (もしくは"SOLID")
PATTERN_DOT (もしくは"DOT")
PATTERN_DASH (もしくは"DASH")
PATTERN_LONGDOT (もしくは"LONGDOT")
```

メソッド

drawPoint(<pos>[,coord])

`drawPoint(<pos>[,coord])`は、カレントビューのベクトル座標位置`<pos>`に事前に設定しておいた描画色とラインパターンを使用して、ポイントを描画します。`coord`パラメータはオプションで（デフォルトでは`SYS_OBJECT`）、以下の値が指定可能です。

```
SYS_WORLD (もしくは "WORLD" もしくは"GLOBAL")
SYS_OBJECT (もしくは"OBJECT" もしくは"LOCAL")
SYS_ICON (もしくは"ICON")
```

drawLine(<pos1>,<pos2>[,coord])

`drawLine(<pos1>,<pos2>[,coord])`は、座標位置`<pos1>`から`<pos2>`まで直線を描画します。

drawTriangle(<pos1>,<pos2>,<pos3>[,coord])

`drawTriangle(<pos1>,<pos2>,<pos3>[,coord])`は、頂点`<pos1>`、`<pos2>`と`<pos3>`を持つ三角形を描画します。

drawCircle(<pos>,rad[,coord])

`drawCircle(<pos>,rad[,coord])`は、中心点`<pos>`、半径`rad`の円を描画します。

drawText(<pos>,text[,coord,[align]])

`drawText(<pos>,text[,coord,[align]])`は、座標位置`<pos>`に指定した文字列を描画します。引数`align`には以下のオプション値を指定可能です。

```
LEFT (もしくは"LEFT")
CENTER (もしくは"CENTER")
RIGHT (もしくは"RIGHT")
```

プラグインAPIは評価の間、状態情報を保持していませんが、これはLScriptでも同様です。つまり、`process()` UDFが呼び出される度に、レイアウトインターフェイス上でオブジェクトに対する描画色を設定しなおす必要があり、描画パターンは常に`PATTERN_SOLID`になってしまうということなのです。

例：

ここにLightWave SDKで提供されている“ **barn** ”サンプルを複製したCustom Object LScriptをご紹介します。

```
  @warnings
  @script custom
  vert = @ <0.0, 0.0, 0.0>, <1.0, 0.0, 0.0>,<1.0, 1.0, 0.0>,<0.5, 1.5, 0.0>,
<0.0, 1.0, 0.0>, <0.0, 0.0,-1.0>,<1.0, 0.0, -1.0>, <1.0, 1.0, -1.0>, <0.5,
1.5,-1.0>,<0.0, 1.0, -1.0> @;
  edge = @ 1, 2, 2, 3, 3, 4, 4, 5, 5, 1, 6, 7, 7, 8, 8, 9, 9, 10, 10, 6,
1, 6, 2, 7, 3, 8, 4, 9, 5, 10, 2, 5, 1, 3 @;
  process: ca
  {
    for(x = 1;x < 31;x += 2)
      ca.drawLine(vert[edge[x]],vert[edge[x+1]]);
    ca.setPattern("dot");
    for(x = 31; x < 35;x += 2)
      ca.drawLine(vert[edge[x]],vert[edge[x+1]]);
  }
```