# LScript v2.8 リリースノート

## 新機能

LightWave(R) v9 における新しいグローバルカメラ設定をサポートするために、 以下の CS コマンドが追加されました ：

GlobalFrameSize()
GlobalResolutionMultiplier()
GlobalPixelAspect()
GlobalApertureHeight()
GlobalMotionBlur()
GlobalParticleBlur()
GlobalBlurLength()
GlobalMaskPosition()

これら新規関数への引数は、 非グローバルな関数の引数に対応しています。 これらのコマンドを使用するとシーンのグローバルのカメラ設定が修正されることになります。 現在選択されているカメラが修正されるわけではありません ( 詳細は LightWave(R) SDK ドキュメントをご覧下さい )。

---

以下の CS コマンドが追加され、 カメラのグローバルリダイレクト設定にアクセスできるようになりました ：

UseGlobalResolution(**<state>**)
解像度設定用のグローバルリダイレクトの <state> を設定します。
値 1 を設定すれば、 グローバルカメラに対して設定されます。
値 0 を設定すると、 カメラ自身の設定が有効になります。

UseGlobalBlur(**<state>**)
モーションブラー設定用のグローバルリダイレクトの <state> を設定します。
値 1 を設定すれば、 グローバルカメラに対して設定されます。
値 0 を設定すると、 カメラ自身の設定が有効になります。

UseGlobalMask(**<state>**)
マスク境界設定用のグローバルリダイレクトの <state> を設定します。
値 1 を設定すれば、 グローバルカメラに対して設定されます。
値 0 を設定すると、 カメラ自身の設定が有効になります
( 注意 ： カメラのマスク境界設定に対し "Use Mask" 設定が有効になっていなければなりません )。

---

LScript の Camera Object Agent はカメラのグローバルリダイレクトをサポートするために、 以下の新規メソッドをサポートするようになりました ：

globalResolution()

解像度用のグローバルリダイレクト設定用の状態を表す整数値を返します。 値 1 が返されれば、 設定が有効 ( カメラの解像度設定は " グローバル " カメラからの設定 ) であるということになります。 値 0 が返されれば、 カメラ自身の解像度設定が有効であるということです。

globalBlur()

モーションブラー用のグローバルリダイレクト設定用の状態を表す整数値を返します。 値 1 が返されれば、 設定が有効 ( カメラのモーションブラー設定は " グローバル " カメラからの設定 ) であるということになります。 値 0 が返されれば、 カメラ自身のモーションブラー設定が有効であるということです。

globalMask()

マスク用のグローバルリダイレクト設定用の状態を表す整数値を返します。 値 1 が返されれば、 設定が有効 ( カメラのマスク設定は " グローバル " カメラからの設定 ) であるということになります。 値 0 が返されれば、 カメラ自身のマスク設定が有効であるということです。

---

LScript における Mesh Object Agent の server() メソッドに対する内部リストに AnimUV、 Camera それに Node ハンドラーとインターフェイスサーバータイプが追加されました。 新規タイプは SERVER_ANIMUV_H、 SERVER_ANIMUV_I、 SERVER_CAMERA_H、 SERVER_CAMERA_I、 SERVER_NODE_H それに SERVER_NODE_I という名称によってスクリプト環境内で定義されています。

---

LScript の Image Object Agent は以下の新規メソッドをサポートするようになりました ：

replace(<imageFile>)

有効なイメージファイルを指定すると、 このメソッドは Agent がプロキシとして提供している現在のイメージに対して、 ファイル内部のイメージデータを入れ替えます。 以下のコードはこの新規メソッドのサンプル例です ( 処理に使用される読み込まれるイメージは "layers.bmp" です ) ：

```
@version 2.8
@warnings

generic
{
	if((replacementName = getfile("Please select replacement im-
age...","*.*",".",true)) == nil)
		return;

	if(replacementName.contains("layers.bmp"))
		return;	// same file that's currently loaded?
```

```
        image = Image("layers.bmp");
        image.replace(replacementName) || error("Replacement failed!");

        info("New image: " + image.filename(0));
    }
```

---

## バグ修正

Envelope Object Agent のコンストラクタはグラフ編集で無効なチャンネルグループを表示してしまう問題があり、 エントリをクリックするとクラッシュを引き起こす危険性がありました。

---

LScript リクエスタパネル内で使用されるアイコンは、 アイコンを使用しているパネルが二度目に開かれたときに内部的に破棄されていたため、 アプリケーションがクラッシュする場合がありました。

# LScript v2.7 リリースノート

## 新機能

スクリプトは **StatusMsg()** CS コマンドを使用できるようになりました (LightWave 3D [8] で追加されたコマンド )。 このコマンドを使用すれば、 スクリプトはレイアウトのメインインターフェイス上にある " ツールチップ " 領域にメッセージを表示できるようになります。 メッセージを表示するだけなので、 ユーザーから中断されることもありません ( 例 : ダイアログを破棄するボタンをクリックされるなど )。

また、 指定するテキストメッセージには、 0.0 ～ 1.0 の間の少数値を波括弧でくくったメタ文字を先頭に追加することが出来ます。 指定した値に到達するとメッセージが表示され、 同時にプログレスバーの背景も描画されます。 例えば :

```
"{.25}Loading object..."
```

このメッセージは 25% の時点でメッセージが表示され、 背景色が描画されることになります。 値を増加させながら **StatusMsg()** を繰り返し呼び出すことにより、 スクリプトはメッセージとともに進捗状況をアニメーションさせることが出来るのです。

```
generic
{
 for(x = 1;x <= 100;x++)
 {
    StatusMsg("{" + x/100.0 + "}This is a test");
    sleep(100);
 }

 StatusMsg("Done!");
}
```

---

LScript はこのバージョンから、 本格的な **Communication Ring** システムのメンバとなっています。 **Communication Ring** とは LightWave 3D [8] に追加されたメカニズムであり、 このシステムを利用することにより、 プラグインは能率よくイベント情報を通信しあうことが出来、 イベント情報とともにデータも通信できるようになります ( Communication Ring システムに関する詳細については LightWave プラグイン SDK のドキュメントをご参照ください )。

このシステムをサポートするため、 新しいコマンドがいくつか追加されました :

comringattach(**<topic>**,**<callback>**)

> 指定した Communication Ring に対するチャンネルを開きます。 各 ComRing は文字列で表される唯一の見出しによって、 識別されます。 リングへと最初にアタッチされるスクライバがリングを作成し、 最後にデ

タッチされるスクライバがリングを破棄します。

それに加え、 他のスクライバによりある特定の ComRing 上でイベントが発生した時点で常に呼び出されるユーザー定義のコールバック関数も定義する必要があります。 このコールバック関数は二つの値を受け取ります。 これらの値は生成されたイベントに対するイベントコード ( 整数値 ) とデータポインタ ( インターナルタイプ ) です。 インターナルデータポインタは **comringencode()** と **comringdecode()** 関数 ( 後述 ) を使用して処理されます。

### comringdetach(<topic>)

スクリプトが ComRing への通信を完了した時点 ( 終了時など ) で各 ComRing へのアタッチ状態も解除するようにしてください。

### comringmsg(<topic>,<code[,<data>])

指定した ComRing 上でイベントを生成します。 このリングに対する他の全てのサブスクライバはイベントの通知を受け取り、 同時にそのイベントに対し処理を行います。 重要な点はイベントはキャッシュされることなく、 通知すると同時に処理されるという点です。 ですから、 リングイベントの処理は出来るだけ早く行わなければなりません。

イベント code はイベントを識別するための単なる整数値です。 この値は完全にユーザー定義の値となります。

data の値はオプションです。 ただし、 他のプラグインに対しデータを転送したい場合にはここで指定した値が **comringencode()** 関数からのみ、転送されることになります。 それ以外のデータはランタイムエラーを引き起こします。

### comringencode(<datalist>,...)

この関数はデータの変数リストを受け取り、 データを他のプラグインからアクセスできるようなインメモリ形式のデータへとエンコードします。 データを処理することにより C プラグインの中で構造体の型変換によりアクセスできるようにしています。 リング上にある他のアクティブな LScript へとデータを送信する場合は、 LScript はデータを抽出するために **comringdecode()** 関数を使用する必要があります。

datalist はターゲットとなるデータタイプを識別するための文字列の配列です。 データタイプは文字列に対しては "**s**"、 整数値は "**i**"、 浮動小数値は "**f**"、 倍精度の浮動小数点数値は "**d**" となります。 文字列の場合、 文字バッファに対するサイズを指定しなくてはなりません。 このサイズの値はデータタイプを表すメタ文字にコロンと整数のサイズ値を付け加えることで、 指定することが出来ます。 例えば 200 倍との文字バッファに文字列を格納したい場合には、 "**s:200**" と指定することになります。

また処理されるデータタイプ用のカウントを指定することも出来ます。 カウント値は番号記号を使用して指定します。 例えば、 処理しなくてはならないデータがそれぞれ 200 バイトのバッファを持つ三つの文字列配列、 整数値、 それに五つの浮動小数点数値から構成されている場合、 datalist は以下のようになります ：

```
"s:200#3","i","f#5"
```

上記の datalist シーケンスは以下の C 構造体としてアクセス可能となる内部メモリへと変換されます ：

```
struct
{
  char    buf[3][200];
  int     ival;
  float   fval[5];
};
```

特定のデータタイプに対しカウント値を指定する場合、 指定しておいた処理されるはずのデータリストの位置に配列もしくは初期ブロックを提供しておく必要があります。 カウント値が指定されている個所に配列や初期ブロックが見つからないとランタイムエラーが発生します。

**comringencode()** から返されるデータ値だけが、 **comringmsg()** 関数の data 値として使用することが可能です。

comringdecode(<datalist>,<data>)

この関数は ComRing イベントに含まれる data からデータを抽出するために使用されます。 指定される datalist は data 値の中に含まれるデータのタイプと順番を定義する必要があります。

エンコードされた値がエレメントのストリームとして返されますので、 そのデータを単一の配列変数へ、 もしくは連想代入を使用して、 それぞれの変数へと要素を抽出していくことが可能です。

"**ComRing**" SDK サンプルプロジェクトではこのメカニズムを実際に実行して紹介しています。 マスター LScript では C コードで書かれた Custom Object プラグインの概観をリアルタイムにコントロールし、 データ通信の二つの手法を図解で紹介しています。

---

LScript v2.7 では URL ファイル参照するという、 かなり面白い新機能のために、フックが提供されています。 現在、 この機能は外部スクリプトコンポーネントが参照される個所 ( ファイルを含む Object Agents など ) で採用されています。

この機能を有効にするためには、 まず **libwww** と呼ばれ配信されている **W3C** コードのプラットフォーム特有の共有ライブラリを取得しなければなりません。

これらのファイル (Windows の場合は約 26 個の *.dll ファイル ) を取得したら、LScripts インストールディレクトリの直下に "**w3c**" ディレクトリをサブディレクトリとして配置します。 例えば、 LightWave のインストール先が "C:\LightWave" の場合には、 この潜在的な LScript 機能を有効にするために **libwww** ファイルは "C:\LightWave\LScripts\W3C" へと配置するようにしてください。

W3C ファイルを適切な場所に配置し、 LScript へと正しく読み込まれると、 スクリプトではローカルファイル名称ではなく、 スクリプトコンポーネント用の URL 参照を指定することが出来るようになります。 例えば HTTP サーバー上で公開されている

Object Agent を参照することが出来るようになるのです ：

```
...
@include "http://www.myobjectagents.com/cooloa.inc"
...
```

さらに、 参照されるインクルードファイルの内部で、 Object Agent 宣言の行は実際の Object Agent バイナリファイルを HTTP サーバーから持ってくるために、 ファイルパスの変わりに URL を指定することが出来るのです ：

```
...
use "http://www.myobjectagents.com/cooloa.oal" as class CoolOA;
...
```

お気に入りのブラウザにおけるウェブページと同様、 このやり方で HTTP サーバーから取得されたファイルはマシン上へとローカルにキャッシュされます (W3C ディレクトリと同じフォルダ )。 ファイルが URL により参照されるたびに、 ローカルファイルデータが HTTP サーバーとチェックされより新しいデータのファイルがローカルで参照されることになります。 こうすることで、 コンポーネントの配信は HTTP サーバー上にあるファイルを更新しさえすれば良くコンポーネントを使用している全ての人々が、 スクリプトを参照するごとに自動的に更新情報を受け取れるようになるのです。

URL ファイル参照は実際にはかなり実験的な機能といえるでしょう。 クロスプラットフォームの使用は保証されてないからです (Windows OS 下でのみ、 テストに成功しています )。

---

Requester Control Object Agent に **size(<w>,<h>)** メソッドが追加されコントロールの幅属性と高さ属性を変更できるようになりました。 これらの属性を変更する場合はまず、 コントロールを隠した状態にしておき、 その後でもう一度、 可視状態へと戻すのが一番良いでしょう。

---

## バグ修正

**matchdirs()** コマンドは Mac 上でサブディレクトリの数を誤ってカウントし、戻り値 **nil** を返していました。

---

**Icon** オブジェクトを使用しているリクエスタは、 リクエスタを破棄する際に、 システムからアイコン情報を適切にクリアしていませんでした。 これにより、 スクリプトを何度も実行させていくうちにシステムが徐々に不安定となり、 アイコン表示がおかしくなったり、 アプリケーションがクラッシュしたりしていました。

# LScript v2.6 パッチ リリースノート

## バグ修正

**generalopts[]** 配列に対するインデックスのバリデーションコードで配列内の最終要素(**[7]**)を除外していました。

---

ランタイム構文解析コードはコンパイル済みスクリプトにおけるコード行に比例して若干、 処理時間に遅れが生じる可能性がありました。 この問題はスクリプトサイズがある一定のサイズ ( 比例 ( 対数 ) 上昇曲線の開始地点 ) になると顕著になり、 それ以降、徐々に読み込み時間も増加していきます。

---

バイナリディスクファイルへの保存 / 復帰により、 '**true**' と '**false**' 定数値は、 ランタイム環境において常に正しく評価されるとは限りませんでした。

---

リクエスタ表示よりも前に行われるタブページ選択は無効となり、 **reqpost()** コードにおいて最初のページで初期化されていました。

---

Displacement Object Agent のデータメンバ '**source**' はプログラムにおける論理フロー内の破綻により隠れていました。

---

リフレッシュコールバック関数 ( コントロールのアクティブなリフレッシュ UDF 内にある間、 コントロールに対する **setvalue()** が呼び出される時点で起動 ) 内で、 無限再帰が行わない用にするための監視メカニズムは、 v2.6 開発環境では破綻してしまう場合がありました。 このバグは修正されました。

---

連想代入に割り当てられた単一要素の戻り値 ( 通常 '**nil**') はエラーメッセージ "**illegal assignment type**" を生成していました。 現在では、 この場合代入する変数は全て初期化された '**nil**' の値を残したままにし、 エラーは出さなくなりました。

---

サブ配列から配列要素への代入において、 例 ：

```
PolysPoints[i][5] = polyinfo(PolyNum[i]);
```

正しい型がマークされていませんでした。 このため、 LScript のごみ箱システムが脆弱となり、 代入文の最後で配列が再利用され、 クリアになっていました。

---

長い間、 破綻をきたしていた **ctlgroup()** 機能が修正されました。

---

スクリプトが処理中に、 描画コールバック関数がレイアウトに対しコントロールを返した場合、 スクリプトの再入問題が生じていました ( 例えば **info()** 関数の呼び出しはスクリプト実行を一時中断してしまうなど )。 このような状況における再入は監視され回避されるようになりました。

---

**info()** ( もしくは **warn()** や **error()**) 関数を使用して、レイアウト上で PointID を表示しようとするとクラッシュしていました。

---

既存のコンパイル済みモデラー用 LScript には **points[]** と **polygons[]** の自動生成において、 問題が生じる可能性があったため、 LScript エンジンにより管理されるようになりました。 この問題が発生した場合、 大抵は "**attempt to access undeclared array ...**" というエラーメッセージが表示されていました。

コンパイル済みスクリプトを読み込む時点でこの状態を捕えるためにコードが追加され、 これら配列に対する参照を修正しました。

---

LScript コンパイラインターフェイスの "**Days until expiration**" と "**Usage count**" には不の値を入力できませんでした。

---

新規 **binary()** 関数は、 2 番目の引数における最大ビットサイズが正数のビット数と等しい場合、 ( 例 ： [**sizeof(int) * 8**])、 ドキュメントに記述されたような戻り値を返していませんでした。

---

Scene Object Agent のデータメンバ **animfilename** に現在のシーン名称が正しくマッピング、 されていませんでした。

---

モデラー LScript コンパイラインターフェイスにある **ecovery password** と **expiration message** コントロールは有効になっていませんでした。

---

**info()/warn()/error()** 関数でベクトル値が正しく表示されていませんでした。

---

モデラーの LScript コンパイラで採用されているテンポラリスクリプト構造は、 作成の際にクリアされておらず、 データメンバ全てを見て意義の状態にしていました。 このためコンパイルとコンパイル済みファイルの結果双方において、 わけのわからないエラーが出ていました。

---

モデラーの **setlayer()** と **setblayer()** 関数は、 ( 配列内に ) 指定したレイヤー番号が特定のカウントよりも超えた場合に、 メモリのオーバーランを引き起こしていました。

---

モデラーの LScript コンパイラはコンパイル中に使用するテンポラリーのスクリプト構造における値を適切に初期化していませんでした。 このためコンパイルと、 コンパイルされたスクリプトの結果双方において、 ランダムに問題が発生していました。

---

オブジェクト ID の変更を自動ハンドリングの処理は **false** 状態に対してチェックしていたため、 状態によっては更新を無視していました。

---

LScript におけるクロスプラットフォーム間のパス変換コードは、 スクリプト内のリテラルパス文字列が使用されているプラットフォームに対し、 より正確に対応できるように再処理を行うようになりました。

---

**reqposition()** 関数は指定された X の値を X 位置と Y 位置双方に対し、 使用していました。

---

ランタイムアプリケーションの配列参照への ID 構築は、 例 ：

```
save_type`current_expr[i] = ...
```

破壊されていました。

---

LScript のプリプロセッサでは連続変数を扱う場合、 基本となる名称にアンダースコア文字 ('_') が含まれていると、 正しい名称を生成することが出来ませんでした。

---

**store()** と **recall()** 関数を使用しているコンパイル済み LScript は、 他のスクリプトを一度実行するまで、 保存されていた値を正しく取得することが出来ない場合がありました。

---

Shader の **process()** 関数においてデバッガを起動すると、 再入問題を引き起こしアプリケーションをクラッシュさせる場合がありました。 この状態はトラップされておりデバッガがアクティブ中に連続して呼び出される **process()** は破棄されます。 デバッガを起動すると Shader スクリプトは破棄された呼び出しに応じて正しい出力結果を得られない ( 例 ： 出力イメージが正しくシェーディングされていないなど ) 傾向にあるという点に注意してください。

---

## 振る舞いの変更

**info()/warn()/error()** 関数でレイアウト内に生成される情報ポップアップは、 今までメッセージの 1 行目には送信したスクリプトの ID を表示していましたが、 これからはスクリプトの名称はウィンドウのタイトルに吹かされ、 メッセージの 1 行目 2 行目ともスクリプトから利用できるようになりました。


---

**ctlpopup()** コントロールは動的にコンテンツを生成できるようになりました。 3 番目の引数として配列の方の値を渡す代わりに、 UDF の名称を指定することにより、 ポップアップはリストが表示されるたびに、 ポップアップの内容を指定するための指定された UDF を参照します。

```
...
ctl[EXPRLST_CTL] = ctlpopup("Expressions",1,"popup_expr_list");
...
```

コールバック UDF は独立した、 もしくは配列参照の形式として、 単数または複数の文字列を返さなくてはなりません。

```
...
popup_expr_list
{
   return(expr_list);  // expr_list[] はスクリプト内の他の場所で構築されます
}
...
```

---

環境内の変化をスクリプトがより柔軟に取り扱えるようにするため、 Scene オブジェクトの情報を管理する Object Agent には新たに二つのメソッドが追加されました。

**isValid()** はシーン内のそれらオブジェクトに対し有効とされているオブジェクト ID を Object Agent 内に保存します。 オブジェクト ID が有効でなくなった時点で、 ブール値 **false** が返されます。

**isOriginal()** はシーン内のあるイベントによって(オブジェクトの読み込みや削除など)基本となるオブジェクト ID が変更されたかどうかを示します。 ブール値 **true** が返されれば Object Agent が作成されてからオブジェクト ID が変更されていることを示します。

---

**pack()** メソッドは連想配列を扱えるようになりました。 '**nil**' の要素を持つキーは全て配列から除去されることになります。

# LScript Editor v1.3 パッチリリースノート

終端に改行文字が来ていないテキストファイルは、 エディタ内でファイルの最終行において正しく終了させることができませんでした。

入力されるテキスト行のサイズが最初に割り当てられるバッファの長さと正確に一致した場合、 要求されているバッファ拡張処理が実行されず、 メモリのオーバーランが発生し、 不規則に不安定となります ( 通常、 断続的なクラッシュの原因となります )。

# Interface Designer v1.3 パッチリリースノート

リクエスタの仕切り線生成のコードで、 **ctlposition** 呼び出しの中で正しい 一時変数を使用していませんでした。

# LScript v2.6 リリースノート

## 新機能

C のプラグインで LScript 汎用リクエスタメカニズムを利用できるようになりました。 このサービスは LCore サブシステムにより提供されており、 プラグインは LScript のリクエスタメカニズムを用いて LScript コードを使用して作成されたユーザーインターフェイスを使用できるようになったのです。 このシステムを使って C プラグインはリクエスタの結果を取得したり ( 例 ： "OK" が押されたのか、 "Cancel" が押されたのかなど )、 初期コントロール値も設定やリクエスタ破棄時における結果を得られるようになりました。

手動でもしくは LSIDE インターフェイスデザイナーにより生成されたリクエスタを実装している LScript コードは、 終端が NULL の文字列配列に保存されます。 この配列がコンパイル用として汎用リクエスタメカニズムへと渡されることになります ：

```
const char *ui_script[] = {
   "options",
   "{",
      "reqbegin(¥"LScript Universal Requester test¥");",

      "c1 = ctlnumber(¥"Number¥",lsur_num);",

      "if(reqpost())",
         "lsur_num = getvalue(c1);",

      "reqend();",
   "}",
   NULL
 };
...

int           err,ok;
char          *messages[10];
LSURFuncs     *lsurFunc;
LWLSURID      script;

double        lsur_num;
void          *vars[1];

lsurFunc = (*global)(LWLSUR_GLOBAL,GFUSE_TRANSIENT);

script = (*lsurFunc->compile)(ui_script,messages,10);
```

汎用リクエスタメカニズムは指定されたスクリプトコードをコンパイルし、 コンパイルされたコードを表す一部不透明なポインタを返します。 スクリプトコードに問題がある場合にはコンパイル中に生成されたメッセージが指定されたレセプタクルに保存されま

す。 問題があると、 これらのメッセージは処理され、 後ほど開放されます ：

```
...
for(x = 0,err = 0;x < 10;x++)
{
   if(!messages[x]) break;

   if(!strncmp(messages[x],"e#",2))
   {
      ++err;
      (*msgFuncs->error)(&messages[x][2],"");
   }

   free(messages[x]);
}

if(err) return(AFUNC_OK);
...
```

生成されたメッセージには重要性を示すメタコードが接頭辞として付けられます。 情報メッセージには "**i#**" が、 警告メッセージには "**w#**"、 エラーメッセージには "**e#**" が付けられます。 コンパイル中にエラーが発生した場合には、 直ちに終了してください ( 返されるスクリプトポインタはいずれの場合には NULL になっているはずです )。

汎用リクエスタメカニズムでは戻り値のパスがどこを指すのかを確認するのと同様初期値の場所も確認してくれます。 しかし正しく実行されるには、 守らなければならない設計要求が幾つかあります。

1．C プラグインと LScript コードの間でデータの交換を行うためには、 コントロールは変数で初期化されていないといけません。
2．データ交換に使用される変数の頭には "**lsur_**" という文字をつけるようにして下さい。 ( 上記コード例参照 )
3．**getvalue()** 関数の呼び出しはある特定の変数で行うようにして下さい。 これは特定の変数に対する戻り値となるコントロールの値を取得し、 C プラグインの中で適切なレセプタクルへと戻すためです。

初期値と戻り値取得用の位置を提供するため、 汎用リクエスタメカニズムに対し (**void ***) のリストが提供されています。 各 (**void ***) は対応付けられているコントロールに対し適切な変数タイプを指しておく必要があります。 例えば **ctlnumber()** ポインタは (**double**) 変数を指していなければなりません。

```
...
lsur_num = 34.54;

vars[0] = (void *)&lsur_num;

ok = (*lsurFunc->post)(script,vars);
...
```

リクエスタが破棄される時点で、 明示的に **getvalue()** の呼び出しを割り当てられていた変数は C プラグインの中で自動的に更新されます。

```
...
```

```
if(ok)
{
   // ユーザーは ″OK″ ボタンを押し 変数は更新されます
   ...
}
...
```

変数は再度変更することが出来、 リクエスタは必要なだけ何度でも表示することが出来ます。 リクエスタを通すときには、 リリース関数へと渡してメモリを開放する必要があります。

```
...
ok = (*lsurFunc->post)(script,vars);
(*lsurFunc->release)(script);

if(ok)
{
...
```

LScript でサポートされているコントロールは全てリクエスタの中で使用出来ますがデータ交換がサポートされているコントロールタイプは以下に記すタイプだけとなっています ：

(char *)
- ctlstring()
- ctltext()
- ctlfilename()
- ctlimage()
- ctlsurface()
- ctlfont()

(integer)
- ctlinteger()
- ctlchoice()
- ctlpopup()
- ctlcheckbox()
- ctlslider()
- ctlminislider()
- ctlstate()

(double)
- ctlnumber()
- ctldistance()
- ctlpercent()
- ctlangle()

(double[3])
- ctlvector()
- ctlcolor()
- ctlrgb()
- ctlhsv()

(LWItemID)

```
        ctlallitems()
        ctlmeshitems()
        ctlboneitems()
        ctlcameraitems()
        ctllightitems()

(LWImageID)
        ctlimageitems()

(LWChannelID)
        ctlchannel()
```

---

LScript では **@save** プラグマを使用して選択される保存モードが新たに追加されました。 **original** モードではスクリプトのフルパス名称をシーンやオブジェクトファイルの中に保存します。 この **original** モードがデフォルトであり、 明示的に選択する必要はありません。

また **relative** モードでもスクリプトを保存することが可能です。 この保存方式ではスクリプトの名称だけをシーンやオブジェクトファイルに保存します。 再読み込みを行うと、 名称を保存されたスクリプトは LScript のデフォルトディレクトリである **¥New-Tek¥LScripts** に存在することになります。 このモードはバッチレンダリングなどを行う場合など、 スクリプトの位置に柔軟性を与えることが出来、 とても便利です。

最後に、 スクリプトは直接シーンファイルやオブジェクトファイルに埋め込んでおくことが可能です。 こうすることで、 スクリプトファイルをどこに置こうと、 またいつどこで読み込み直そうとも常に利用可能な状態にすることが出来ます。

```
@save   original
@save   relative
@save   embedded
```

これらの設定を有効にするためには、 初回はまずスクリプトをディスクファイルから起動しなくてはなりません。 その後のシーンやオブジェクトへの保存 / 読み込みでは、常にその設定を維持してくれることになります。 このモードを無効にするためには、スクリプトを無効にし、 シーンやオブジェクトを再保存する必要があります。

埋め込み型スクリプトに対してデバッグは行わないでください。 これらの保存モードを起動する前にデバッグを行うようにして下さい。

---

統合データタイプに対し **contains()** メソッドが新たに利用できるようになりました。 このメソッドは指定されたデータタイプをインデックス可能なタイプ ( 配列や文字列など ) を通じて検索し、 そのタイプや型が存在するかを判断するメソッドです。 指定された値に対し、 存在する場合には **true**、 しない場合には **false** のブール値を返します。

```
...
str = "Now is the time";
info(str.contains("is the"));   // true (1) を表示
info(str.contains("Bob"));      // false (0) を表示
...
```

```
...
t = @15.87,"Blowfish",<1,9,0.5>@;
info(t.contains(<1,9,0.5>));    // true (1) を表示
info(t.contains(15.87));        // true (1) を表示
info(t.contains("Blow"));       // false (0) を表示
...
```

---

**keys()** メソッドは連想配列に対し適用し、 配列内に含まれるキー全てを抽出することが出来るようになりました。 これによりスクリプトはリニア形式における連想配列内のデータ要素全てにアクセスできるようになります。

---

スクリプトリクエスタで新規コントロールタイプ **ctlviewport()** が利用できるようになりました。 このコントロールは **ctlinfo()** と似ていますが、 より広いカンバス領域をビューポートとして機能します。 キャンバスはビューポートウィンドウよりもさらに広く、水平 / 垂直スクロールバーを持ちます。

**ctlviewport()** への引数は **ctlinfo()** と同じであり、 仮想キャンバスの大きさをレポートする追加コールバックに対し保存を行います。 サイズ関数は常に再描画関数が呼び出される前に即座に呼び出されるため、 キャンバスを状態に合うように動的にリサイズすることが出来ます。

```
c1 = ctlviewport(200,200,"vp_redraw","vp_size");
```

このサイズコールバック関数はビューポート ( このサイズは初期関数呼び出し内で設定されます ) がスキャンするキャンバスの幅と高さを返します。 サイズ関数は例えば、 固定されたサイズ、 またはキャンバス上に配置されたオブジェクトの境界領域を計算します。 関数は大きさを適用しているコントロール ID を受け取ります。

```
vp_size: ctl
{
   return(800,600);
}
```

キャンバス上ではビューポートのサイズに関わりなく描画が行われます。 描画関数は一度にキャンバス全体が可視状態であると想定して行わなくてはなりません。 描画関数を適用するコントロール ID を受け取ります。

```
vp_redraw: ctl
{
   drawbox(<80,80,80>,0,0,800,600);

   drawbox(<200,200,200>,50,50,20,20);
   drawbox(<0,200,0>,750,450,20,20);
}
```

これに加え、 Control Object Agent クラスは **xoffset** と yoffset という二つのデータメンバをエクスポートするようにないました。 この二つのデータメンバは現在ビューポートコントロールに対し全てのタイプで定数 0 となっています。 データメンバの値は

現在のコントロールのビューポートの X( 左 ) と Y( 上 ) が入っています。 キャンバス上のマウスイベントの仮想位置などを正確に計算する場合などに使用出来ます。

```
reqmousedown: mouse_x, mouse_y, ctrl
{
   x = mouse_x - ctrl.x + ctrl.xoffset;
   y = mouse_y - ctrl.y + ctrl.yoffset;
   ...
}
```

また仮想キャンバス上の特定位置におけるビューポートの位置に値を割り当てることも可能です。 指定可能な値の範囲は水平方向においては 0( ゼロ ) ～ (canvass_width - viewport_width)、 垂直方向においては 0( ゼロ ) ～ (canvass_height - viewport_height) となっています。 この範囲を越えた場合、 LScript は割り当てられた値がこの範囲に収まるように自動的に切り抜きます。

---

リクエスタメカニズムは新しいプリミティブ描画関数 **drawcircle()**、 **drawelipse()**、 **drawfillcircle()** それに **drawfillelipse()** の四つを提供するようになりました。 名前が示している通り、 最初の二つの関数は塗りつぶされていない円形を、 後の二つは塗りつぶした状態の円形を、 パネル上または他の有効な描画領域 ( 例 ： **ctlviewport()** ) に描画します。

**drawcircle()** は四つの引数 **(1) color (2) X center point (3) Y center point** それに **(4) radius** を、 **drawfillcircle()** は五つの引数 **(1) border color (2) fill color (3) X center point (4) Y center point** そして **(5) radius** を受け取ります。 数値は全て整数値に解釈されます。

**drawelipse()** は五つの引数 **(1) color (2) X center point (3) Y center point(4) X radius** それに **(5) Y radius** を、 **drawfillelipse()** は六つの引数 **(1) border color (2) fill color (3) X center point (4) Y center point (5) X radius** それに **(6) Y radius** を受け取ります。 数値は全て整数値に解釈されます。

---

新規 **reqposition()** コマンドを使用することでリクエスタパネルの位置が設定 / 配置が可能になりました。 コマンドは二つのオプション引数を受け取ります。 この引数はパネルに対する画面 X と Y の位置を特定します。 これらの値は **reqbegin()** と **reqend()** の間であれば常時設定可能です。 この関数が呼び出されたときにパネルが開かれている状態であればパネルの位置は即座に変更されます。

引数の有無に関わらず、 呼び出された場合は常に **reqposition()** は関数呼び出し時点におけるパネルの現在位置を返します。 これによりスクリプトは画面上のパネル位置を独自に管理できるようになります ( LScript は v2.5 から自動的にこの処理を行ってきましたが、 保存できるパネルの個数に制限があります )。

```
generic
{
   reqbegin("Position Test");
   ...
   req_x = recall("req_x",0);
   req_y = recall("req_y",0);
```

```
        reqposition(req_x,req_y);
        ...
        return if !reqpost();
        ...
        (x,y) = reqposition();
        store("req_x",x);
        store("req_y",y);

        reqend();
        ...
    }
```

**reqposition()** は開いているパネルを即座に更新しますので、 表示中の位置はリアルタイムに処理可能です ：

```
    req_x,req_y;

    generic
    {
        reqbegin("Position Test");
        ...
        return if !reqpost("idle",500);
        ...
        reqend();
        ...
    }

    idle
    {
        ++req_x;
        ++req_y;
        reqposition(req_x,req_y);
    }
```

---

事前定義のスクリプト関数 **save()** と **load()** に渡される I/O Object Agent に対し新規 I/O 関数が利用できるようになりました。 これら新規メソッドは OBJECT I/O モードにおいて数値データタイプをとり扱っている場合に、 さらに細かい精度を提供します。

**readDouble()** はソースファイルから **double**(2 倍)の数値(一般的には 8 バイト)でデータを抜き出します。 **writeDouble()** は同じサイズの数値を保存します。 既存の **readNumber()**/**writeNumber()** は引き続き**float**(浮動小数点数値)サイズの数値を使用する演算を処理します。

**readShort()**/**writeShort()** は短いサイズの数値 ( 一般的には 2 バイト ) で動作します。**readInt()**/**writeInt()** 関数は引き続き整数サイズの値 ( 一般的には 4 バイト ) で動作します。

---

新規 File Object Agent メソッドが利用可能になり、 数値データタイプを扱う場合に、より精密さを提供できるようになりました。

**readDouble()** はソースファイルから **double**(2倍)の数値(一般的には8バイト)でデータを抜き出します。 **writeDouble()** は同じサイズの数値を保存します。 既存の **readNumber()**/**writeNumber()** は引き続き**float**(浮動小数点数値)サイズの数値を使用する演算を処理します。

**readShort()**/**writeShort()** は短いサイズの数値 ( 一般的には 2 バイト ) で動作します。**readInt()**/**writeInt()** 関数は引き続き整数サイズの値 ( 一般的には 4 バイト ) で動作します。

バイトオーダーの管理用に全 File Object Agent バイナリモードメソッドが受け入れる新規オプション引数に関する詳細は振る舞いの変更の章をご覧下さい。

---

三つの頂点マップタイプ、 ウェイト、 テクスチャまたはモーフの頂点マップとインデックス値 0 を **VMap()** コンストラクタを指定することによりモデラーで現在選択されているマップを取得出来るようになりました ：

```
vmap = VMap(VMWEIGHT,0) || error("Select a Weight map so I have something to do!");
```

---

新規レイアウト関数 **visitnodes()** が利用できるようになりました。 この関数はオブジェクトの親子階層を通して反復処理を簡易化してくれます。

関数には二つの引数が必要となります。 最初の引数には参照 Object Agent が渡されます。 渡される Object Agent のタイプは子アイテムを持つことの出来るオブジェクトタイプ Mesh や Light、 Camera、 Bone となります。

二番目の引数は階層内で子オブジェクトがそれぞれ検出された時点で、 LScript により呼び出されるスクリプト内の UDF を指定するための文字列です。 UDF は二つの引数 ( 親にあたる Object Agent 子にあたる Object Agent) を受け取ります。 親アイテムが複数の子アイテムを管理している場合、 複数の呼び出しが同じ親 ID により行われるという点に注意してください。

```
generic
{
   visitnodes(Mesh("MasterObject"),"process_node");
}

process_node: parent, child
{
   info(parent.name," -> ",child.name);
}
```

---

Object Agent に新規メソッド **keyExists()** が追加されました。 このメソッドは指定したタイムインデックスにおいてキーフレーム情報を含むオブジェクトの全チャンネルを識別するために使用出来ます。 指定された時刻におけるキーフレーム情報を含む各チャンネルが、 このメソッドにより返されます。

```
generic
```

```
{
   if((channels = Mesh("Null").keyExists(2.0)) != nil)
   {
      foreach(c,channels)
         info("Null." + c.name);
   }
}
```

---

文字列の形式で、 整数値を対応するバイナリへと変換するための **binary()** 関数が追加されました。 この関数は整数値とオプションとなるサイズの値 ( どのプラットフォーム上でも最大値は [**sizeof(int) * 8**] ビットとなります ) を受け取ります。 この関数はビットの個数を明示的に指定しない限りは、 先頭のゼロを除去します ：

```
generic
{
   info(binary(32));      // displays "100000"
   info(binary(32,32));   // displays "00000000000000000000000000100000"
   info(binary(130,8));   // displays "10000010"
   info(binary(130,10));  // displays "0010000010"
}
```

---

LScript において **Glyph** という新しい Object Agent クラスが定義されました。このクラスはカラーイメージのコンテナとなっています。 このカラーイメージはリクエスタや **ctlinfo()**、 **ctlviewport()** などディスプレイコンテキスト内に描画されるカラーアイコンとしての目的をもつイメージです。

Glyph は Glyph コンストラクタにファイル名称を引数として提供することにより、ディスク上のファイルから構成されます。 ディスク上のタイルを指定する代わりに、バイナリブロックに埋め込まれているファイル名称でもイメージデータソースとして指定することが可能です ：

```
...
cursor_img = Glyph(cursorGlyph);
...
@data cursorGlyph 500
000 000 002 000 000 000 000 000 000 000 000 000 012 000 012 000 024 000 080
080
...
```

何らかの理由により glyph 構築に失敗した場合には 'nil' が返されます。

現在、 Glyph Object Agent はパブリックメソッドを何もエクスポートしません。 以下のパブリックデータメンバが利用可能です ：

```
w          glyph イメージの幅 ( 読み込み専用 )
h          glyph イメージの高さ ( 読み込み専用 )
```

**pixel[col,row]** **glyph** 画像のピクセルカラーデータ配列 ( 初期モードでは書き込み専用、デュアルモードでは読み書き両用 )

一旦構築されれば、 glyph は新規 **drawglyph()** 関数を使用してディスプレイコンテキストへと描画されます。 この関数には Glyph Object Agent と共に glyph が描画されるコンテキスト内部における X と Y の位置が渡されます ：

```
...
// カーソル glyph を描画
drawglyph(cursor_img,
    cursor_pos.x - integer(cursor_img.w / 2),
    cursor_pos.y - integer(cursor_img.h / 2));
...
```

さらに二つのオプション引数が **glyph()** コンストラクタには用意されています。 デフォルトでは Glyph は最高速度スピードでディスプレイコンテキストに描画できるような方式で生成されます。 この描画速度と相反するのがイメージ内の個々のピクセルに直接アクセスできる能力です。 **glyph()** コンストラクタに指定可能な 2 番目の引数には Glyph ( デュアルモード ) 内のイメージデータに対しピクセルベースにアクセスする必要があるということを LScript に示すブール値のフラグが指定できます。 例えば、 Image Filter データ上に Glyph を重ねようとしている場合など (Image Filter データは正当なディスプレイコンテキストではないため Glyph の高速モードでは直接その上に描画することは不可能なのです )、 このモードを指定する必要があります。

3 番目のオプション引数は Glyph 描画時において使用される透明なカラーマスクの値です。 このようなマスクの使用においてはイメージデータへピクセルごとにアクセスが必要となるため、 透明なマスクを指定すると自動的に Glyph オブジェクトにおいてこのモードが有効になるのです。

Glyph イメージデータはホストアプリケーションにおいて有効である Image Loader プラグインを全てサポートしているため、 バイナリブロックを通してイメージデータをスクリプト内へと埋め込む場合にはファイルのタイプのヒントを指定する必要が出てきます。 デフォルトでは、 LScript はバイナリブロック内のイメージデータを処理する前に ".tga" 拡張子をつけます。 Targa 形式ではないイメージデータでもうまく動作する場合もありますが、 そうではない場合もあります。 ファイルタイプのヒントは適切なファイルの拡張子を指定し、 バイナリブロックの名称にはアンダースコア文字 (_) で区切り付け加えます。

```
@data cursorGlyph_jpg 500
000 000 002 000 000 000 000 000 000 000 000 000 012 000 012 000 024 000 080
080
080 080 080 080 080 080 080 080 080 080 080 080 080 080 080 080 080 080 080
080
...
085 069 086 073 083 073 079 078 045 088 070 073 076 069 046 000
@end
```

---

ユーザー定義関数におけるローカル変数は、 変数名の頭に "**st_**" をつけることにより、スタティック ( 静的 ) 変数に変更することが出来ます ( 静的変数とは、 関数が何度呼び出されても、 値を保持している変数です )。

```
generic
{
```

```
    for(x = 0;x < 5;x++)
        docount();
}

docount
{
    if(!st_value)
        st_value = 0;

    info(++st_value);
}
```

---

モデラー LScript に新たに五つのコマンドシーケンス関数が追加されました ：

revert([<filename>])

現在のオブジェクトを、 指定したディスクファイルに保存されているメッシュ情報へと復帰します。 ファイルが存在しない場合には、 LScript はオブジェクトファイルの現在のファイル名称を使用して復帰させようとします ( ファイルが未保存の場合には、 当然ですがエラーが返されます )。

selectvmap(<type>,<name>)

編集用に特定の VMap のタイプと名称を選択します。 <type> には **Morph**、 **Spot**、 **Weight**、 **Subpatch weight** または **Texture UV** を指定します。 これらのタイプは VMap() Object Agent コンストラクタに渡されるのと同じ **VMAP** タイプを使用して指定します。

meshedit(<name>)

指定した名称を持つ MeshEdit クラスのプラグインを呼び出します。

smoothscale(<distance>)

選択しているメッシュに対し <distance> で指定した値でスムーズシフトを実行します。

changepart(<name>)

現在選択されているポリゴンに対しパート名称を ( 再度 ) 割り当てます。

---

Mesh Object Agent は以下の二つのパブリックメソッドをエクスポートするようになりました ：

layerName(<layernum>)

オブジェクトの指定したレイヤーに割り当てられているレイヤー名称を返します。 レイヤーに対し特定の名称が割り当てられていない場合には、 **'nil'** を返します。

layerVisible(**<layernum>**)

指定したレイヤーが可視状態かどうかを示すブール (**true** または **false**) 値を返します。

---

新しい文字分析メソッドがシステムに追加されました。 これらのメソッドはそれぞれ ANSI C 規格に直接対応しており、 整数値や文字列における最初の文字だけが認識されている文字列の値などに適用可能です。

どのメソッドも引数を受け取らず、 ブール値 (**true**/**false**) を返します。

isPrint()

印刷可能な文字かどうかを判別します。

isAlpha()

文字構成が英字クラス ( 例えば a-z または A-Z ) であるかどうかを判別します。

isAlnum()

文字構成が英数字クラスであるかどうかを判別します。

isAscii()

文字構成が ASCII 文字セットかどうかを判別します。

isCntrl()

文字構成が制御文字セットかどうかを判別します。

isDigit()

文字構成が数値文字セット ( 例 0-9) かどうかを判別します。

isPunct()

文字構成が句読文字 ( 例 スペース文字ではない印刷可能な任意の文字、または **isAlnum()** で **true** が返される文字 ) かどうかを判別します。

isSpace()

文字が空白 ( 例 空白文字、 または 0X09 ～ 0X0d 間の文字 ) かどうかを判別します。

isUpper()

文字構成が大文字英字セットかどうかを判別します。

isLower()

文字構成が小文字英字セットかどうかを判別します。

isXDigit()

文字構成が 16 進数文字セット ( 例 "0123456789ABCDEF" のうちの一つ ) かどうかを判別します。

---

既存の LScript **log()** 関数は指定された値の自然対数 ( 基数が **e** ) を計算しています。 これは基数 **10** を求める機能に誤解されがちなので、 新しい関数 **log10()** を追加しました。

```
generic
{
   x = log(23);
   y = log10(23);
   z = log(23) / log(10);  // log10(23) と 同様
}
```

---

LScript 言語に新しい乗算命令子 "**^^**" が追加されました。 この命令子は **pow()** 関数の代わりに言語内で直接使用することが可能です。

```
t = 5^^2;      // 25
i = 10;
info(i ^^ 3);   // 1000 と表示
```

---

Image Filter LScripts には **overlayglyph()** というイメージデータキャッシュ上で Glyph Object Agent と混合できる関数が追加されました。 この関数には Glyph Object Agent と Glyph がイメージキャッシュ上に張り込まれる X と Y オフセット位置を渡します。 引数の 4 番目には イメージキャッシュ上に合成される Glyph イメージの透明度の値を表すオプション引数を指定します。 この値の範囲は 0.0 ～ 1.0 で、 0.0 の場合は全く透明度がなく ( デフォルトはこの状態 )、 1.0 の場合には Glyph は 100%の透明度を持ちます (Glyph は不可視状態になります )。

```
bug;

create
{
   bug = Glyph(vt4000,true);
   setdesc("Bug");
}

process: width, height, frame, starttime, endtime
{
   overlayglyph(bug,1,1);
   overlayglyph(bug,1,height - bug.h,0.25);
   overlayglyph(bug,width - bug.w,height - bug.h,0.5);
```

```
    overlayglyph(bug,width - bug.w,1,0.75);
    overlayglyph(bug,(width - bug.w) / 2,(height - bug.h) / 2,0.25);
}

@data vt4000 67800
000 000 002 000 000 000 000 000 000 000 000 000 188 000 120 000 024 000 026
026
026 025 025 025 025 025 025 025 025 025 026 026 026 026 026 026 025 025 025
025
...
000 000 000 000 000 000 084 082 085 069 086 073 083 073 079 078 045 088 070
073
076 069 046 000
@end
```

結果 :

---

Channnel Object Agent は新たに **parent** というデータメンバを提供するようになりました。 このデータメンバにはチャンネルが属している LightWave の Object Agent が含まれています。

---

Scene Object Agent の **generalopts[]** 配列に、 インターフェイスの " 自動キー " ボタンの現在の状態を示す 7 番目の要素 **[7]** が加えられました。 この値は " オン " の場合にはブール値 **true** が、 " オフ " の場合は **false** が返されます。

---

Scene Object Agent に以下のデータメンバが追加されました ：

alertlevel

レイアウトにおける警告レベルの設定状態を示します。 **ALERT_BEGINNER**、 **ALERT_INTERMEDIATE** または **ALERT_EXPERT** です。

boxthreshold

現在のバウンディングボックスのしきい値設定の値を示す整数値です。

autokeycreate

レイアウトにおける警告キー作成設定を示しています。 **AKC_OFF**、 **AKC_MODIFIED** もしくは **AKC_ALL** のいずれかの値になります。

numthreads

レンダリング中に分散されるスレッド数です。

animfilename

現在選択されているアニメーションファイルの名称です。 設定されていない場合には '**nil**' になります。

rgbprefix

RGB ファイル保存の名称です。 設定されていない場合は '**nil**' になります。

alphaprefix

Alpha チャンネルのファイル保存の名称です。 設定されていない場合は 'nil' になります。

---

LightWave の Object Agents は **axislocks[]** と呼ばれる配列をエクスポートするようになりました。 この配列には九つの要素が含まれており、 三つの要素が一組となってオブジェクトの位置、 回転、 スケールチャンネルに対応するロック状態状態を表す値が入っています。 各 3 個 1 組の要素は変形カテゴリに対する適切な軸に対応しており特定のチャンネル / 軸要素のロック状態を示すブール値 (**true** の場合はロック ) が入っています。

Icon Object Agent が LScript に追加されました。 このオブジェクトタイプはピクセルのビットマップパターンを構築 / 構成するために設計されています。 これらのピクセルのパターンは完全にユーザー定義の文字キャラクタを表現します。 Icon キャラクタはリクエスタパネルがアクティブな間のみ、 有効となります。

各文字の最大値は幅 16 ピクセルに縦 14 ピクセルとなっています。 文字は配列内に配置された文字列により定義されます。 各文字列の幅は最大値を越えてはいけませんが、各配列内にある文字列の個数は最大の高さを越しても構いません。 各文字列内において、 ピリオド文字 ('.') は 0 ピクセルの値 ( オフ ) を示しており、 ピリオド以外の文字は 1 ピクセルの値 ( オン ) を表しています。 各配列は **Icon()** コンストラクタへと渡され、黒白の文字イメージを構築 / 保存することになります。

```
folder_icon = @ "..1111.....",
          ".1....111111....",
          ".1.........1...",
          ".1.........1...",
          ".1...1111111111.",
          ".1..1.........1",
          ".1.1.........1.",
          ".11.........1..",
          ".111111111111..."
         @;


eyes_icon = @ ".................",
         ".................",
         "...mmm...mmm....",
         "......m.m.......",
         "....mm...mm.....",
         "...m..m.m..m....",
         "..m.mm.m.mm.m...",
         "..m.m..m.m..m...",
         "..m....m....m...",
         "...m..m.m..m....",
         "....mm...mm.....",
         ".................",
         "................."
        @;

@define FOLDER     1
@define EYES       2

generic
{
   icon[FOLDER] = Icon(folder_icon);
   icon[EYES] = Icon(eyes_icon);

   reqbegin("Testing Icons");

   c1 = ctlbutton("Open File " + icon[FOLDER],75,"bfunc1");
   c2 = ctlstring(icon[EYES].asStr() + " Browse","testing");
```

```
    reqpost();
}

bfunc1
{
    ...
}
```

こうすればアイコンが文字列の中に埋め込まれ、 LScript が処理を行いイメージ ( ボタンやコントロールのラベル文字、 リストボックスのエントリーなど ) を表示するのです。


---

Custom Object の flags() 関数は SCHEMA に加え、 以下の値も返すようになりました ：

VPINDEX

view データメンバ内の値はタイプではなくビューポート番号に相当する値を示します。

NODEPTH

Z 深度に関係なく他の全ての OpenGL 描画の前面にオブジェクトの描画が行われます。

---

新規コマンド **getdir()** は定数値 **SCRIPTSDIR** もしくは文字列 "**scripts**" を認識しローカルシステム上における **NewTek¥LScripts** のインストールディレクトリを指すパスを返します。

---

プラットフォーム特有のパスの区切り文字を返す **getsep()** 関数が追加されました。

```
...
info(getsep());    // OS によって ″¥¥″, ″/″ または ″:″ を表示
...
```

---

Displacement Access Object Agent には Displacement LScripts の関数 **process()** が提供されていますが、 新たに **point** データメンバをエクスポートするようになりました。 このデータメンバには現在処理を行っているメッシュポイントに対する Point Object Agent が入っています。

---

LScript は 16 進数形式の数値をサポートするようになりました。 これらの数値は文字列の先頭に "**0x**" が付けられており、 1-8 桁の 16 進数値が記入可能です (A-F、 a-f、 0-9)。

```
...
t = 5 * 0xa0;      // 5 * 160 と同じ
...
```

また **integer()** 関数が拡張され、 文字列形式内における 16 進数値を認識し処理できるようになりました。

```
...
t = integer("0x0a");                    // t には 10 が 入ります
t = integer("Now is the 0xF000 time");     // t には 61440 が 入ります
...
```

---

LScript はバイナリ形式の数値をサポートするようになりました。 これらの数値は文字列の先頭に "**0b**" が付けられており、 1-32 桁のバイナリの数値が含まれています (0 もしくは 1)。

```
...
t = 0b1001;        // t には 9 が入ります
...
```

また **integer()** 関数が拡張され、 文字列形式内におけるバイナリの数値を認識し処理できるようになりました。

```
...
t = integer("0b110");                    // t には 6 が入ります
t = integer("Now is the 0b10000 time");    // t には 16 が入ります
...
```

---

新規追加された関数 **lscriptVersion()** はスクリプトを実行している LScript システムのバージョン情報を返します。 この関数では 4 つの値を次の順番で返します。 1 番目は LScript バージョンを表す文字列です (LScript 実行時にスクリプト選択ファイルダイアログのタイトルバーに表示されるバージョンと同じです )。 2 番目はバージョンのメジャーコンポーネントを表す整数値、 3 番目はマイナーコンポーネントを表す整数値です。 最後はバージョンのパッチレベルを表す整数値です。

---

Layout LScript コンパイラはコンパイル済みスクリプトの生成において新しいモードが追加されました。 この新規コンパイルタイプは "**library**" ( ライブラリ ) といい、 関数 ( 使用されないものも含む可能性あり ) の集合体を目的とされており、 単一のプラグインアーキテクチャには連結されていません。 一度コンパイルされるとライブラリコマンドを使用することにより、 この "**library**" スクリプトを他の LScript から使用することが可能です。

"**library**" ファイル内部で定義されている関数は LScript に元々組み込まれているかのように、 スクリプトから参照することが可能になります。

```
@version 2.6
@warnings
@script generic

// ライブラリ 'functions.lsc' には gimmStringFrom() 関数が含まれています

library "functions.lsc";   // パスがありませんのでファイルは ¥NewTek¥LScripts にあるものと判断します

generic
{
   t = 104;
   info(gimmeStringFrom(t));
}
```

---

アクティブなリクエスタパネル上でキーボードの動きを途中で捕えるための新規コールバック関数 **reqkeyboard()** が追加されました。 この関数は押されたキーを表す単一の引数を受け取ります。 このキーがシステムにより、 さらに処理を続行するか否かを表す **false** または **true** の値を返します。

```
@version 2.6
@warnings
@script generic

generic
{
   reqbegin("Testing reqkeyboard()");

   c1 = ctlstring("String","value");

   if(reqpost())
      info("You pressed Ok");
   else
      info("You pressed Cancel");

   reqend();
}

reqkeyboard: key
```

```
{
   if(key == 13)      // エンターキー
   {
      reqabort(true);
      return(true);
   }
   else if(key == 27)  // エスケープキー
   {
      reqabort();
      return(true);
   }

   return(false);
}
```

以下の事前定義定数値がキーイベント処理用に追加されました ：

```
REQKB_F1          REQKB_KB0          REQKB_KP0
REQKB_F2          REQKB_KB1          REQKB_KP1
REQKB_F3          REQKB_KB2          REQKB_KP2
REQKB_F4          REQKB_KB3          REQKB_KP3
REQKB_F5          REQKB_KB4          REQKB_KP4
REQKB_F6          REQKB_KB5          REQKB_KP5
REQKB_F7          REQKB_KB6          REQKB_KP6
REQKB_F8          REQKB_KB7          REQKB_KP7
REQKB_F9          REQKB_KB8          REQKB_KP8
REQKB_F10          REQKB_KB9          REQKB_KP9
REQKB_F11
REQKB_F12          REQKB_ALT          REQKB_RETURN
            REQKB_SHIFT          REQKB_INSERT
REQKB_LEFT          REQKB_CTRL          REQKB_HOME
REQKB_RIGHT                        REQKB_END
REQKB_UP          REQKB_DELETE          REQKB_PAGEUP
REQKB_DOWN          REQKB_HELP          REQKB_PAGEDOWN
```

---

新規 **indexOf()** メソッドが文字列、 線形配列、 それにバイナリのブロックデータタイプに適用できるようになりました。 このメソッドはこれらのインデックス可能なタイプ内部にデータを配置するのに使用できます。 このメソッドは特定のデータが検出された箇所のインデックスを返します。 検出されない場合は、 0 の値が返されます。

文字列の場合、 指定された検索用の値は表示可能なキャラクタとして扱われます ：

```
...
s = "val1:15:3.14";
ndx = s.indexOf(':');   // 5 が返ります
...
```

線形配列に対しては、 指定された値との照合に成功したと判断されるためには、 要素のタイプと値双方が一致していなくてはなりません。 全てのタイプに対しこの検索がサポートされているわけではありません。

バイナリデータにおける検索は指定された整数値を符号なしキャラクタとして扱うため、 範囲は 0 ～ 255 となります。

デフォルトでは検索は初期インデックスオフセット値 1 から始まります。 検索値の前に整数のオフセット値を指定することにより検索開始位置を指定することも可能です。

```
...
s = "val1:15:3.14";
ndx = s.indexOf(':');          // 5 を 返します
ndx = s.indexOf(ndx + 1,':');  // 8 を 返します
...
```

---

Surface Object Agent に **getValue()** と対となるメソッド **setValue()** が追加されました。 チャンネル名称と適切な値を指定することにより、 サーフェイスのチャンネルの値を変更することができます。

```
...
(obj) = Scene().getSelect();
(firstsrf) = Surface(obj);
srf = Surface(firstsrf);

srf.setValue(SURFCOLR,<0,255,255>);

translucency = srf.getValue(SURFTRNL);
srf.setValue(SURFTRNL,translucency * 2);
...
```

---

**getValue()** がサポートされているチャンネルは、 Image Object Agent を呼び出しているチャンネル以外全て **setValue()** を使用して変更可能です。

---

LScript のプリプロセッサは新しいコンパイル時のプラグマ **@sequence** を認識するようになりました。 このプラグマを使うと、 連続して値を増減させている名称の集合体を定義することが出来ます。 このプラグマタイプは C の **enum** 関数と似ています。

シーケンシャルな名称の集合は {} 括弧で囲みます ：

```
...
@sequence { ... }
...
```

集合体の中の名称はコンマで区切られます ：

```
...
@sequence { VIEWPORT_CTL, COPY_CTL, PASTE_CTL }
...
```

バイナリデータ行のように、 シーケンス集合体におけるエントリは複数行に及ぶことが

出来ます ：

```
...
@sequence { VIEWPORT_CTL,
        COPY_CTL,
        PASTE_CTL }
...
```

デフォルトでは値は 1 から始まり、 その後一つずつ値を増やしていきます。 上記例では、 **VIEWPORT_CTL** が **1** となり、 **COPY_CTL** は **2** というように続いていきます。 連続値は集合体エントリに対し新しい値を割り当てることで、 任意の箇所から書き換えていくことが出来ます ：

```
...
@sequence { VIEWPORT_CTL,
        COPY_CTL = 5,
        PASTE_CTL }
...
```

この場合、 **VIEWPORT_CTL** は **1** となりますが、 **COPY_CTL** は **5**、 **PASTE_CTL** には **6** が割り当てられます。

増加値もまた、 コロンを使用して新しい増加値と連続値を区切ることにより、 指定することが可能です ：

```
...
@sequence { VIEWPORT_CTL = 1:2,
        COPY_CTL,
        PASTE_CTL }
...
```

このコードでは、 **VIEWPORT_CTL** は依然 **1** のままですが、 **COPY_CTL** は **3**、 **PASTE_CTL** は **5** となります。

一旦宣言されたら、 これら連続した値を持つエントリは **@define** を使用してた定義された値の代わりにスクリプト内の任意の箇所で使用することが可能になります。

---

## 振る舞いの変更

LScript の実行メカニズムはグローバルなコンテキストを全て除去するように再設計されました。 スクリプトのコンテキストは LightWave プラグインで標準であるように関数から関数へと渡されます。

---

LScript の実行メカニズムを再設計したことにより、ユーザー定義の Object Agent と DLL 関数に対するインターフェイスが **LSFunc** 構造体において不透明なコンテキストポインタを含むように変更されました。 この不透明なポインタは構造体内部で定義される LScript 関数の全ての呼び出しに含まれなければなりません。 インターフェイスのバー

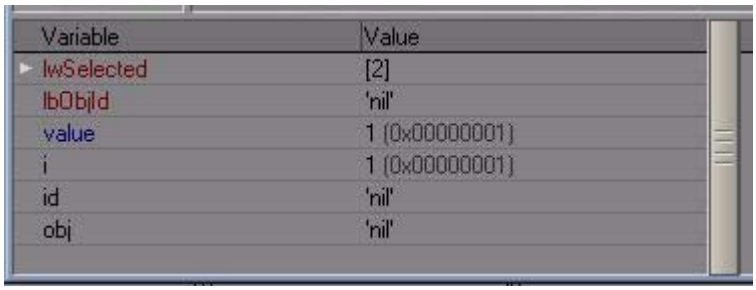
ジョンは 1.4 に上がり、新しい Object Agent や DLL 開発はこの値を LScript に対して返し、 正しい構造体のポインタがコールバック関数に提供されているかどうかを確かめなくてはなりません。

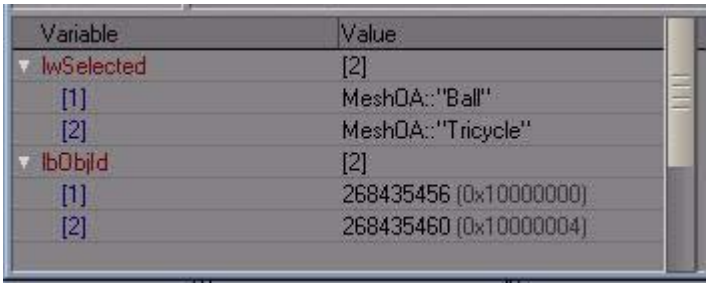
既存のバイナリ Object Agent または DLL ファイルは以前 (1.3) のインターフェイスバージョンにおける LScript の関数を継続していますが、 可能であれば Object Agent コードは v1.4 インターフェイスに更新されることを強くお勧めします。 v1.3 インターフェイスは LScript メモリマネージャーにより統治されることはなくなります。

以降の開発には更新された LScript header file をダウンロードし、 ご使用してください。

---

LScript Debugger の変数監視システムが再設計されました。 現在の関数の引数とローカル変数は関数が有効である限り、 自動的に監視ウィンドウにに表示されるようになります。 各新規関数に入る時点 ( もしくは返される時点 ) で、 その変数は以前の関数のものと置き換えられます。 この振る舞いは一般的なデバッガに一致しています。 情報ディスプレイもまた大幅に拡張されています。




グローバル変数は自動的に監視ウィンドウに追加されなくなりました。 グローバル変数を監視ウィンドウに追加するためには、 左マウスボタンでスクリプト内のハイライト上になっている変数名称をダブルクリックしデバッグメニュー ( または F3 キー ) から "Add Watch" を選択してください。 選択された文字がスクリプト内部のグローバル変数と一致した場合、 監視ウィンドウに追加され、 関数を出たり入ったり移動するにつれて監視ウィンドウ内に留まります。

---

モデラーの **new()** コマンドが成功すると、 作成された新規オブジェクトに対する Mesh Object Agent を返すようになりました。

```
main
{
   m1 = Mesh(0);
   info(m1.id);

   m2 = new();
   if(!m2.isInt())
      info(m2.id);
}
```

---

**Mesh()** コンストラクタは インデックスの値が範囲外である場合に、 エラーメッセージを出しスクリプトの実行を停止するのではなく 'nil' を返すようになりました。

---

LScript は自動的に任意の永続性を持つ Object Agent リファレンス ( 例えばスクリプト内にある変数や配列内に保存されている ) をレイアウトで変更されたときは常に、 新しいオブジェクト ID へと自動的に更新するようになりました。 これによりスクリプトによる介入が要求されることなく、 変更を意識せずにすむようになります。

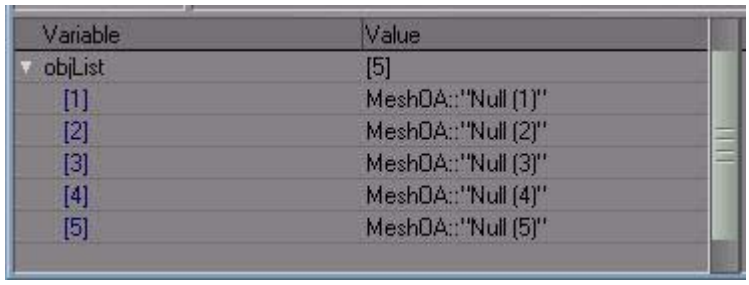
変更が発生したことを認知するプログラムを書かない限り、 スクリプトにおいて多少の混乱をきたす振る舞いになりえる場合もあります。 例えば、 処理用の配列に Mesh Object Agent を保存するスクリプトは以下のようになります ：

```
...
curObj = Mesh();
while(curObj)
{
   objList[++x] = curObj;
   curObj = curObj.next();
}
...
```

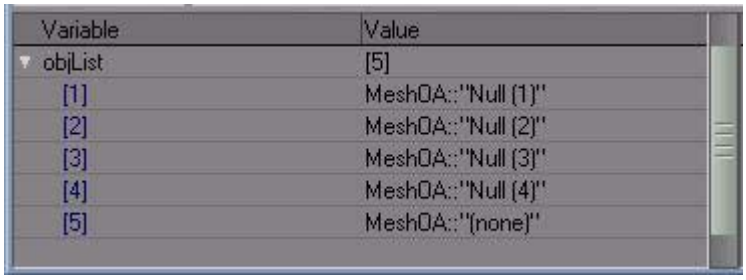
スクリプトの後半で、 配列に Agent が保存されているある特定のオブジェクトが Command Sequence 関数である **ClearSelected()** 関数を使用し、 レイアウトのシーンからクリアされます ：

```
...
objList[2].select();
ClearSelected();
...
```

**ClearSelected()** が呼び出されると、 レイアウトはシーンから選択されたオブジェクトを削除します。 このアクションは一般的に現在のシーンにおける既存オブジェクトの内部 ID が全て変更されるという連鎖反応を引き起こしてしまいます。 このイベントは LScript の透明性によりハンドルされています。 しかし削除前に配列のコンテンツを監視することで結果を見ることが出来ます ：




そしてその後は ：




シーン内の変化により、 まるで配列自自身が変更したかのように見えるのがわかりますね。 実際、 各 MeshObject Agent の基調をなすオブジェクト ID は更新されており、これらの ( 新しい ) 対応するオブジェクト名称がデバッガの監視ウィンドウに表示されるようになりました。

配列内にある 5 番目の要素は、 シーン内にある "(**5**)" ヌルオブジェクトを参照していましたが、 現在では ID 自身が "(**none**)" と表示されています。 これは配列にあるオブジェクト ID が現在のシーンにおいてもはや有効ではないということを示しています。

このタイプの Object Agents を保存している場合には、 このように振舞う可能性があると把握しておく必要があります。 上記の例から Object Agents を配列に保存しているシーンからオブジェクトを削除すると、 配列を処理するコード内における従属の順番を避ける必要があります。

---

Mouse 関数は単一の Object Agent 引数を受け取り、 利用可能なパラメータ全てを単一のパッケージへと包括出来るようになりました。 この Object Agent は以下のデータメンバーをエクスポートします ：

| | |
|---|---|
| ctl | イベント内で呼び出される Control Object Agent ( ない場合には "nil") |
| x | イベント時の X 座標値 |
| y | イベント時の Y 座標値 |
| button | イベントが引き起こされたマウスボタン。 イベント : 1=LMB, 2=MMB, 3=RMB |
| count | クリック回数 |
| keys[3] | 有効な修飾キー |

Object Agent は現在パブリックなメソッドをエクスポートしません。

```
...
reqmousemove: md
{
if(md.ctl)
{
   vp_x = md.x - md.ctl.x + md.ctl.xoffset;
   ...
```

この変更により古い形式のマウスハンドリング関数を使用しているコンパイル済み LScript では問題が生じてしまいます。 それら古いスクリプトはこのリリースにより正しく動作するように必ず更新し再コンパイルしてください。

---

バイナリファイルモードで取り扱う File Object Agent のメソッドがブール値のオプション引数を受け取るようになりました。 この引数は読み込まれた数値をバイトオーダーでスワップするかどうかを指定します。 デフォルトではバイトオーダーはファイルから読み込まれたままのものとなります。 ブール値 **true** を渡すと以前返された値をスワップすることになります。

---

定数値 **LINUX** が追加され **platform()** などの関数により返されるようになりました。 またプリプロセッサの条件ビルド **@if** システム使用に対し判別が可能です。 この値に対し、どのテストも **true** を返すようであれば、 スクリプトは Screamernet で走らせていることを示します (**runningUnder()** は **SCREAMERNET** を返すよう保証されています)。

---

複数行のコメント内にプラグマ指示語が現れているような状況を扱えるようにプロプロセッサはスクリプト上で複数パス処理を実行するようになりました ：

```
/*
@define FOO  100
@autoerror
*/
```

---

イメージを読み込み処理、 **ctlimage()** や **glyph()** などで行われるような処理はホストアプリケーション ( レイアウトもしくはモデラー ) において有効な Image I/O プラグインを通して行われるようになりました。 つまり Targa 形式に比べイメージファイルフォーマットが大幅に扱えるようになったということです。

---

**ctlimage()** 関数は指定したイメージファイルの読み込みが失敗した場合には '**nil**' を返すようになりました。 通常は要求された ImageLoader プラグインがアプリケーションにインストールされていないことを示すものです ( 指定しれたイメージファイルが単に存在しないことによるエラーであれば、 ユーザーに対しエラーメッセージが表示されます )。 Targa 以外のイメージファイルを使用しているスクリプトはこの戻り値をチェックしてコントロールが適正に作成されているかを確かめる必要があります。

---

**ChannelGroup()** コンストラクタは引数として ChannelGroup Object Agent を受け取るようになりました。 この引数を渡すと、 指定されたチャンネルグループの下にある一番目のサブグループを返しますが指定されたチャンネルグループの下にあるサブグループ全てを検索するためには二つの引数が必要となります。 2 番目の ChannelGroup Object Agent は " 現在の " チャンネルを示すとみなし、 以降に定義されるサブグループ全てが返されます ( サブグループがもはや存在しない場合には '**nil**' が返されます )。

---

文字列の値が指定されている場合、 **setvalue()** 関数はまずはじめに **ctlpopup()** コントロールに対するリストの値を一致しているかを確かめに行きます。

---

不連続 UV の値の処理を可能にするために、 数種類の VMap Object Agent のメソッドが引数リストを拡張しました。

isMapped(**<point>**[,**<polygon>**])

**isMapped()** は 2 番目のオプション引数を受け取るようになりました。この引数は指定したポリゴンに対し不連続 UV 値がチェックされるポリゴンに対する引数です。

getValue(**<point>**[,**<polygon>**][,**<index>**])

**getValue()** は 2 番目の引数を処理します。 この引数は不連続 UV の値を検索するポリゴンを指定しています。 <index> の値もまた特定の値インデックスに対する検索の限度を指定するものです。

setValue(**<point>**,**<value>**|**<array>**[,**<polygon>**][,**<index>**]) ( モデラーのみ )

**<polygon>** は **setValue()** 引数リスト内に含まれます。 これはポリゴンの各頂点に対し割り当てられた ( 不連続 )UV 値です。

---

**loadimage()** と **clearimage()** 関数はグローバル関数となり、 レイアウトもしくはモデラーのどちらでも使用できるようになりました。

---

Mesh()、 Camera()、 Light() それに Image() コンストラクタはスキャンコードが拡張されました。 オブジェクト名称が指定されると、 これらのコンストラクタはアルファベットの大文字小文字に関係なく番最初に見つかった一致するオブジェクト名称を検索します。 ただし、 システム内に同じ名称で複数個のオブジェクトが存在する場合には、 検索を成功させるために指定された名称は一意な名称をつけておく必要があります。

例えば、 "Cow.lwo" という名称のオブジェクトがアプリケーション内部に読み込まれているとします。 以下のコードはオブジェクトの検索照合に成功します ：

```
...
obj = Mesh("cow");
...
```

ただし、 その後で "COW.lwo" という名称のオブジェクトが読み込まれると、 アプリケーションの内部に "Cow" と "COW" というオブジェクトが存在することになります。先ほどのコードでは検索に失敗 ('nil' が返されます ) してしまいますので、 読み込まれたオブジェクトのうち、 一つのオブジェクトとだけ明瞭に検索一致するように変更する必要があります ：

```
...
obj = Mesh("Cow");
...
```

アプリケーションからスクリプトに対し返されるオブジェクト名称にはそれぞれの大文字小文字の設定が含まれているため、 使用時には常に正しいオブジェクト名称との検索照合を行います。

---

プラグマ指示子を認識するためには全て最初の行からはじめなくてはなりません。 初期ブロックのトークン ('@') が行 ( 一番目以外の行 ) 上の一番目の文字として現れる場合に、 混乱を招かないためです。

---

**ctlbutton()** 関数は 4 番目のオプション引数を受け取るようになりました。この引数はボタンに割り当てられているコールバック関数へと渡される引数リストを構成しています。引数リストは二重引用符で囲まれており ( 例えば文字列として渡す )、 一つ以上の引数がコンマで各々区切られています。

```
...
reqbegin("Testing");
c1 = ctlbutton("press me",70,"press_me","10,r1,r2");
reqpost();
...
```

指定された引数は定数値 ( 数値もしくは二重引用符で囲まれた文字列 ) や、 **ctlbutton()** を呼び出す関数へのグローバルもしくはローカルへのリファレンス用の変数名称の場合もあります。

```
r1 = "bob";

generic
{
   r2 = "hood";

   reqbegin("Testing");
   c1 = ctlbutton("press me",70,"press_me","10,r1,r2");
   reqpost();
   reqend();
}
```

```
press_me: arg1, arg2, arg3
{
   info(arg1," ",arg2," ",arg3);
}
```

---

**ctlslider()** 関数は 5 番目に整数値のオプション引数を受け取るようになりました。 この引数はスライダコントロールの絶対は場を指定するために使用されます。 この幅はスライダ自身の幅とは独立しています。

---

LScript Debugger の監視ウィンドウとメッセージウィンドウはインタラクティブにリサイズできるようになりました。 これらのウィンドウの上部の開いた領域のところをクリックしドラッグすると事前に定義された限界内部において領域を貸し出来る限りリサイズ可能です。

---

**count()** メソッドは文字列タイプのオブジェクトが適用された場合に、 文字列引数を提供出来るようになりました。 その場合、 呼び出し側へはキャラクタの発生回数が整数値として返されます。

```
...
s = "1:2:5:38";
info(s.count(':'));   // 3と 表示します
...
```

---

**asStr()** メソッドは線形配列に対しても適用できるようになりました。 サポートされるデータタイプは連結され単一の文字列へと変換されます。 **'nil'** の値を持つ要素は無視されます。 文字列に追加される値を区切るために使用されるオプションの文字列値を指定することも可能です。

```
...
a[1] = "val1";
a[2] = 15;
a[3] = nil;
a[4] = 3.14;
info(a.asStr(':'));    // "val1:15:3.14" と表示されます
...
```

---

モデラー LScript においての **CommandInput()** 関数が使用可能になりました。 モデラー特有の CommandSequence 関数を呼び出したり、 LScript 内部において特別に指定されていない双方のアプリケーションでの共通コマンドも使用できるようになりました。

```
...
```

```
CommandInput("Surf_SetWindowPos 100 50");
...
```

Command Sequence 関数に対応している任意の利用可能な LScript 特有の関数仕様もスクリプト内部において引数のタイプと値を提供することにより継続して使用することが出来ます。

---

## バグ修正

実行メカニズムの再設計したことにより、 Procedural Texture の LScript はレンダリングが複数スレッドで処理される場合に正しく機能するようになりました。

---

未保存のオブジェクトから Mesh(0) コンストラクタの呼び出しが行われた場合に既にディスクに保存されているワークスペース内のオブジェクトに対する Agent が返されていました。

---

Mesh(0) の呼び出しはレイアウト上で発生する不正なインデックス処理を引き起こしてしまうコードを起動していました。 問題が発生すると、 大概の場合、 代わりに不正なオブジェクトが現れていました。

---

**requpdate()** 関数は引数が何も指定されていない場合、 パネルのリフレッシュを行いませんでした。

---

**setvalue()** 関数は空の文字列値を正しく受け取っていませんでした。

---

想定していた値を返さない関数は LScript の内部スタックを破壊し、 比較時にその値を直接使用するとクラッシュを引き起こしていました。 例 ：

```
if(NotDone() == true)
...

NotDone
{
   if(x == 0)
      return(true);

   // このパスからは何も返りません
}
```

---

**server()** メソッドには正反対の状態テストが含まれていました。 これにより何かの値を返す代わりに 'nil' が返されていました。

---

**makecone()** コマンドは浮動小数点数値の引数を整数値へと不正に変換していました。

---

**ctlinfo()** コントロールがタブページに割り当てられている場合、 正しくレンダリングされていませんでした。

---

**SelectItem()** はボーン名称参照を適切に処理していませんでした。

---

前置 / 後置加算と前置 / 後置減算子が配列の要素に対して直接適用されている場合に正しく機能していませんでした。

---

コメント内部にあるプラグマ指示子がプリプロセッサにより正しく素通りされていませんでした。

---

**setvalue()** と **getvalue()** 関数は Tab コントロールに関しては適切に動作していませんでした。

---

**foreach()** 演算子は Mesh 以外の Object Agent タイプでは機能しませんでした。

---

**foreach()** 演算子は処理用に提供されている配列内の Layout Object Agent を認識しませんでした。

---

**filecopy()** 関数にはソースファイルのコンテンツを破壊してしまうバグがありました。

---

まれなケースとして反復に使用される変数の箇所の **foreach()** 演算子が他の変数に割り当てるため LScript のメモリ管理によってその後再利用されるメモリポインタを保持してしまう問題が発生しました。 この場合、 扶養情報の収集により関数のローカル変数が再利用される時点でクラッシュする可能性がありました。 クロスリンクの状態では二つの変数のうち、 一つが開放されると、 もう一つの変数は再利用される時点で定義されていない状態のままで残ってしまいます。

---

ライトの種類を定義する定数値が幾つかから抜け落ちていました。

---

Polygon Object Agent の **points[]** データメンバはそのオブジェクトタイプに正しく割り当てられていないため、 アクセス不能になっていました。

---

スクリプトに返される前に **getWorldRotation()** メソッドの測量が正しく度数へと変換されていませんでした。

---

**ShadowColor()** 関数は定義されている正しい引数の個数を保持していませんでした。

---

**ctlangle()** コントロールを初期化するために数値全体が使用される場合、 コントロールの値はスクロールがどの方向にドラッグされているかに限らず増分処理だけを行っていました。

---

利用可能なスロットが全て埋まっている場合、 リクエスタパネルの位置メカニズムにおいてメモリのオーバーランが発生しました。 パネルの位置は現在のリクエスタにより置き換えられる最も古いリクエスタで変化しますが、 リスト内の実際の位置は使用されませんでした。 このオーバーランはとりわけ明白なバグでありマウスポインタがリクエスタパネルに侵入すると同時にアプリケーションがクラッシュを起こします。

---

他のモーダルリクエスタから生成されたモーダルなリクエスタの中で発生するコントロールイベントによりどちらが現在アクティブなパネルなのかが混乱してしまっていました。 一般的には子パネルがアクセス不能になり、 マウスの動きによりクラッシュを起こす可能性があります。

---

**fogColor()** メソッドはカラーの値を全てベクトルの X 要素に対し割り当てていました。

---

**ctlchannel()** コードはリストボックス内で特定の初期チャンネルの配置を正しく行えないという微妙な問題がありました。 これにより不安定な状態となり、 その結果クラッシュを引き起こしていました。

---

File の行カウント処理で最終行に改行コードが含まれていない場合、 正確にカウントできていませんでした。

---

Queue Object Agents は LScript の不要部分整理システムによる更新が適切になされておらず、 よくない状態になる可能性がありました ( クラッシュやフリーズなど )。

---

**split()** 関数は指定されたパスのドライブコンポーネントが 8 文字位以上の場合は常にメモリのオーバーランを引き起こしていました。 PC ドライブの指名子は大概の場合は二文字で構成されているため、 それほど問題は生じなかったのですが、 Mac でこの問題は発生する確率が高くありました。

---

Command Sequence の幾つかの関数 (**TargetItem()** や **ParentItem()** など ) ではボーンを有効なターゲットとして認識できていませんでした。

---

**setvalue()** 関数を使用したリストボックス選択の設定における問題を修正しました。

---

リストボックスの更新と **requpdate()** 関数における問題を修正しました。

---

Particle System Object Agent の所有権はシステム内で明示的に認識されておらず、それに適用されている変数が不要情報整理システムにより構成された時点で、 Object Agent ( と基幹をなすパーティクルシステム ) が破壊される恐れがありました。

---

**meshedit()** 関数が **SelectVMap** コマンドを呼び出していました。

---

リクエスタの区切り文字が適切に消去されていませんでした。

---

**ctlactive()** メカニズムは複合化されたコントロールの全エレメントを扱っていませんでした。

---

標準の C のファイルハンドル関数を使用する **@insert** メカニズムが正しく機能していませんでした。 代わりにプリプロセッサはメモリに挿入する処理を行うよう再設計されました。

---

LScript のデバッガはスクリプトメッセージを正しくキャプチャしていませんでした。

---

LScript デバッガは監視ウィンドウとメッセージウィンドウにおけるエントリのハンドルに問題がありクラッシュする原因となっていました。

---

リクエスタのマウス関数はノンモーダルパネルの場合、 パネルのクライアント領域全体を変換していませんでした。

---

LScript の構成モジュール **store()** や **recall()** など、 関数からデータを管理する責任があるモジュールには、 キーをスキャンする際に C の **strncmp()** 関数を使用すると曖昧な問題を潜在的に秘めていました。 この問題は Windows 以外のプラットフォームにおいて発生していました。

# LSIDE Editor v1.3 リリースノート

## 新機能

LightWave の LCore サブシステム ( とりわけ LScript を構成するシステム ) は新しいインターフェイスを持つようになりました。 これにより LScript はリンクしているアプリケーション内部に埋め込まれるようになりました ( 現在、 LightWave アプリケーションの内部のみがこの恩恵にあずかっています )。

LSIDE Editor は LCore の新しい 埋め込み型 LScript メカニズムを使用する初めてのクライアントです。 "Tools" メニューの下にある新しいオプション "Macro..." は Editor マクロの管理システムに対しインターフェイスを提供します。 このメニューエントリを選択することでウィンドウが開き、 Editor マクロの管理を行います。 マクロファイルには拡張子 "els" がつきます。

3 番目のポップアップメニューが Editor のインターフェイス右上に追加されました。 このメニューには現在読み込まれているマクロが列挙されており、 実行用に使用されます。

Editor は二種類のマクロ、 一回きりのマクロとキーベースのマクロをサポートしています。 一回きりのマクロは LightWave における Generic クラスのスクリプトのような働きをします。 呼び出された時点で一度だけ実行し、 終了した段階で処理が完了します、 キーベースのマクロ ( ここでは以降 " フィルター " マクロと称します ) は Master や Motion などの LightWave のスクリプトと同じく、 起動した時点からキーストロークの処理用に呼び出されるか、 明示的に非アクティブにするまでバックグラウンドにてアイドル状態を保ちます。

Editor のマクロシステムは LScript によって管理されているため、 LScript から提供される LightWave 非特有の関数もまた、 埋め込み型 LScript を使用して他のアプリケーションでも利用出来るようになります。 これにより言語機能だけでなく、 特定の総括ビルトイン関数に拡張されます。

ただし埋め込み型 LScript を採用しているアプリケーションは、 そのスクリプト内で使用するためのコンテキスト特有の独自の関数セットを定義することが出来ます。 以下は LSIDE Editor により定義されたコンテキスト関数のリストです。

**<doc> newDoc([<filename>])**

この関数はエディタ内に新規ドキュメントを作成します。 有効なファイル名称が指定されれば、 新規ドキュメントにそのファイルが読み込まれます。 そうでない場合空のドキュメントが生成されます。

**<doc> currentDoc([<doc>])**

現在のドキュメントへのハンドルが返されます。 ハンドルが引数として指定されている場合、 エディタ内でそのドキュメントがフォーカスになります。

**<doc>** nextDoc(**<doc>**)

この関数はシステムにおいて指定されたドキュメントハンドルの後に続く ( 読み込み順序に従って ) ドキュメントへのハンドルを返します。

saveDoc(**<doc>**[,**<filename>**])

この関数は指定したドキュメントハンドルがさすドキュメントを保存します。

**<integer>** lineCount(**<doc>**)

指定されたドキュメントハンドルにおける行数を返します。

**<string>** getLine(**<doc>**,**<line>**)

指定したドキュメントハンドルにおける特定の行番号のコンテンツを返します。

**<color[]>** getColor(**<doc>**,**<line>**)

指定されたドキュメントハンドルにおける特定の行番号のカラー表示を返します。 返された値はベクトル形式での RGB カラー値の配列となっています。

addLine(**<doc>**,**<string>**[,**<after>**])

指定した文字列を指定したドキュメントハンドルに追加します。 オプションで行番号が指定されている場合、 ドキュメントにおける指定した行番号の後に文字列行が追加されます。 そうでない場合には、 ドキュメントの終端に追加されることになります。

deleteLine(**<doc>**,**<line>**)

指定したドキュメントハンドルにおける指定した番号の行を削除します。

deleteSelection(**<doc>**)

指定したドキュメントハンドルにおいて現在選択されている文字を削除します。

replaceLine(**<doc>**,**<line>**,**<string>**)

指定した番号の行を指定したテキストで置き換えます。

replaceColor(**<doc>**,**<line>**,**<color[]>**)

指定した行番号のカラーコンテンツを指定したカラーベクトルの配列で置き換えます。

markLine(**<doc>**,**<line>**)

エディタにおいて指定した行番号の左側にマーカーを配置するようにドキュメントの表現を変更します。 検索ダイアログにおける "Mark All Matches" 設定と同一です。

**<Boolean>** isDirty(**<doc>**)

指定されたドキュメントハンドルが修正されているかどうか ( 例えば保存の必要があるかなど ) をチェックします。

**<type>** docType(**<doc>**[,**<type>**])

ドキュメントハンドルの種類を返します。 返されるタイプとしては **DOC_SCRIPT**, **DOC_CSOURCE**, **DOC_CHEADER**, もしくは **DOC_OTHER** があります。、 またドキュメントタイプを変更するためのタイプ指示子を提供することも出来ます ( タイプ指示子は構文ハイライト用にとても重要な役割を果たします )。

**<string>** docName(**<doc>**[,**<fullpath>**])

指定されたドキュメントの名称を返します。 ドキュメントのまだ名称がつけられていない場合 ( つまりディスクに未保存の場合 ) は、 **'nil'** が返されます。 二番目の引数としてブール値 **true** を指定すると、 ドキュメントのフルパスを取得することも出来ます。

highlightSyntax(**<doc>**)

指定されたドキュメントに対し構文ハイライトを有効にします。

(**<row>**,**<col>**) getCursor(**<doc>**)

指定されたドキュメントにおける現在のカーソル行と位置を返します。

setCursor(**<doc>**,**<row>**,**<col>**)

指定したドキュメントにおいて指定した行と列の箇所にカーソルを配置します。

**<Boolean>** selection(**<doc>**)

指定されたドキュメントにおいて、 現在有効なテキスト選択がなされているかを問います。

(**<row>**,**<col>**) getSelect(**<doc>**)

選択されている行と列の位置を返します。 選択状態が有効であれば、 この範囲はカーソルの動きによって増減します。 したがって、 この関数から返される値は選択の " アンカーポイント " となります。

(**<start>**,**<end>**) getLineSelect(**<doc>**,**<line>**)

ドキュメントの現在の選択に含まれる指定した行の開始列と終了列を返します。 この範囲は行全体を含むことも出来ますし、 カーソル位置と選択アンカーの位置にしたがって一部分だけを取得することも出来ます。

setSelect(<doc>,<row>,<col>)

指定したドキュメントにおける選択アンカーポイントの行 / 列位置を確立します。

<Boolean> selected(<doc>,<line>)

指定したドキュメントにおいて指定した行がドキュメントの現在選択の中に含まれているかどうか ( 全体か一部か ) を問います。

<integer> countChar(<doc>,<char>[,<include>])

指定されたドキュメントをスキャンし、 指定された文字列の発生回数をカウントします。 オプションの **include** 引数がブール値 **true** であれば文字列 ( つまり引用符で囲まれた文字列 ) もまた、 スキャンされカウントされます。

message(<string>)

エディターのインターフェイスのメッセージ領域においてテキストメッセージを表示します。

<Boolean> inString(<doc>,<row>,<col>)

ドキュメントにおいて指定した行と列の位置がキャラクタ文字列の内部にあるかどうか ( 引用符に囲まれている文字列 ) を示します。

<Boolean> overwrite(<doc>[,<overwrite>])

指定したドキュメントのブール値 **true** が読み書き両用を示す読み込み専用の状態値を返します。 ドキュメントの読み込み専用状態はオプション引数としてブール値 **false** を指定した場合に設定されます。

<string[]> funcNames(<doc>)

指定したドキュメント内部に現在する関数の名称を返します。 ドキュメントのタイプは必ず **DOC_SCRIPT** であり、 関数が定義されていなければなりません。 そうでない場合には '**nil**' が返されます。

<Boolean> funcIsCollapsed(<doc>,<name>)

指定したドキュメントにおいて名称付けられた関数の破損状況を返します。 ブール値 **true** は破損状態を示します。

toggleCollapse(<doc>,<name>)

指定したドキュメントにおいて名称付けられている関数から認識される関数本体を拡張または破損します。 この関数を呼び出す前に **funcIsCollapsed()** を使用して関数の状態を事前に調べてください。

<integer> funcLine(<doc>,<name>)

指定したドキュメントにおける名称付けられた関数の行数を返します。 関数が存在しない場合には、 'nil' が返されます。

以下の関数がマクロをフィルターするためのみに利用可能です ：

replaceKey(<key>)

現在処理中のキーを指定されたキーへと代えます。 これから行う全てのキーフィルターはオリジナルのキーを呼び出す場合に、 入れ替えたキーを受け取ることになります。 また、 この新しいキーがエディタへも渡されることにもなります。

マクロタイプは全て "macro" と呼ばれる中央のエントリポイントを持っています。 これは Editor マクロが開始を実行する場所であり、 一回きりのマクロの場合には、 macro() 関数が終了すると処理が完了します。

```
macro
{
...
}
```

フィルタマクロは filter() と呼ばれる追加のユーザー定義関数を提供しています。 この関数には二つの目的があります。 ひとつは、 この関数の存在によりキーベースのフィルターとしてマクロを認識されます。 次に LightWave LScript の flags() 関数のように振る舞うことで有効になっている単数もしくは複数の文字を返します。 後者の機能に対しては、 filter() 関数は要求されたキーの値にマッピングされている文字列と整数値の組み合わせを返します ：

```
filter
{
      return("¥ta",13);    // タブと小文字の 'a' を引っ掛け、 キーを返します
}
```

filter() 関数はマクロが起動するたびに Editor のマクロシステムから呼び出されます。 つまりマクロが起動時にまず呼び出される時 ( 前回のセッションでマクロを有効にしていた場合 ) またはインターフェイスから明示的にマクロを呼び出すときに発生します。 このため必要であればマクロの状態を初期化する良い位置を提供します。

```
filter
{
   pastPartials = nil;  // 新規実行用にリセット
   return("¥ta",13);    // タブと小文字の 'a' を引っ掛け、 キーを返します
}
```

またフィルターの macro() 関数は、 アクティブなドキュメントと処理されているキーを表す一つもしくは二つのオプション引数を受け取ります。 フィルターマクロの中で単一のキーがトラップされている箇所では、 2 番目の引数は必要ありません。 また currentDoc() 関数がアクティブなドキュメントに対するハンドルを返しますので引数リストとしてこの値を得ることは重複する作業となってしまいます。 ただしキーが複数割り込む箇所やフィルタマクロがすべて印字可能文字をトラップしている箇所などには、 これらの

引数は役に立ちます ( 前者の場合はイベントを引き起こすキーを決定し、 後者の場合は実行速度のためです )。 指定されたキーの引数はイベントキーの整数値となります。

```
filter
{
   pastPartials = nil;  // 新規実行用にリセット
   return("¥ta",13);    // タブと小文字の 'a' を引っ掛け、 キーを返します
}

macro: doc, key
{
   switch(key)
   {
      case '¥t':     // タブキー
         ...
         break;

      case 'a':
         ...
         break;

      case 13:       // リターンキー
         ...
         break;
   }
}
```

LScript プラグマの幾つかは埋め込み型 LScript により実行されるスクリプトによって使用可能です。 例えば **@name** プラグマは Editor マクロから提示可能 ( そうでなければならない ) でありユーザーインターフェイス上でリスト内にあるマクロの名称を設定するのに使用されます。

```
@name Sort Ascending

macro
{
...
}
```

LightWave で直接実行される他の LScript プラグマは (**@fpdepth**, **@define**, **@strict** など ) もまた Editor ( 埋め込み型 LScript) マクロ内に表示されます。

以下は Editor のマクロシステム開発中に作成されたマクロの例です。 実際的な例もありますし、 提供されたコンテキスト関数の使い方の例としてのみ、 役に立つマクロもあります ：

表 1：

| 名称 | タイプ | 解説 |
|---|---|---|
| autosave.els | フィルタ | 指定したキーストロークの後に現在のドキュメントを保存する |

表 1：

| 名称 | タイプ | 解説 |
|---|---|---|
| a_to_A.els | フィルタ | replaceKey() 関数を使用して全種類の小文字 'a' を大文字化する |
| cf_template.els | 一回きり | 新規ドキュメントを作成し Channel Filter のテンプレートを配置するシンプルなスクリプト |
| changecase.els | 一回きり | ドキュメントにおける文字の大文字小文字を修正する。現在の選択されているものに対し LScript のリクエスタメカニズムを採用する |
| color_e.els | 一回きり | 現在行における全ての 'e' または 'E' の色を変更する |
| disablecode.els | 一回きり | 現在選択している各行の始端に単一行コメントの指示子である (//) を配置する |
| enablecode.els | 一回きり | 選択中の各行の始端から単一行コメントの指示子を取り除く |
| keycomplete.els | フィルタ | タブキーが押された時点で特定のエントリからキーワードを完了する |
| reformat.els | 一回きり | より読みやすくするためにスクリプトを再フォーマットする |
| smartindent.els | フィルタ | 見やすくするためにコードに基づいたインデントを自動化する |
| sort.els | 一回きり | ドキュメントの選択行を昇順に並び替える |

---

ドキュメントテキストはマウス上のマウスホイールを使用してスクロールが出来るようになりました。 またホイールで加速スクローリングも可能です。 ホイールのクリックよりもより短い時間で多くの行をスクロールできるようになります。

---

Editor においてファイルを開くのダイアログが複数選択できるようになりました。

---

## 振る舞いの変更

Editor のアンドゥシステムが再設計されました。 これによりアンドゥメカニズムが正しく機能するようになっただけではなく、 実行中の圧縮コードの追加も同じ量のメモリを消費しながらそれぞれのドキュメントでより多くのアンドゥ実行が可能になりました。

アンドゥシステムではアンドゥイベント回数を特定数で制限するのではなく、 累積されるメモリサイズ ( このリリースではドキュメントに付き 500K) で制限するようになりました。 この制限を超える変更をドキュメントに施した時点で、 アンドゥイベントの累積メモリ消費が再び指定された最大値以下になるまで一番古いアンドゥイベントから破棄されていきます。 実際に破棄されたイベント数はこのアンドゥプールにおける特定のメモリフットプリントに依存します。

---

ドキュメントに対し置き換えのメカニズムが適用されている場合、 選択範囲内部に直接機能するようになりました。

# エクスプレッション

## 新機能

エクスプレッションに LScript を埋め込むという方式を採用することにより、 ユーザーは LightWave のエクスプレッションで使用する独自の関数を書くことが出来るようになりました。

エクスプレッション UDF はビルトインのエクスプレッション関数と同様に使用します。関数へ渡されるパラメータはシンプルなデータタイプ、 つまり文字列や数値、 ベクトルに限られています。 エクスプレッションがこれらデータタイプのうちの一つを評価する間は、 引数としても使用可能です。

エクスプレッション UDF は \NewTek\LScripts ディレクトリの中にある "**expressions**" という独自のディレクトリ内に保存されます。 関数のデフォルトライブラリはこのディレクトリの中にある "**library.ls**" に保持されます。 この関数ライブラリは LightWave が初期化された段階でエクスプレッションエンジンの中に自動的に読み込まれ、 その結果ライブラリに定義された関数は、 関数を参照するエクスプレッションやエクスプレッション UDF から利用出来るようになります。

さらに、 それぞれのエクスプレッション UDF は同じディレクトリの中に、 独自のファイルを保存することが可能です。 UDF を含むファイル名称は参照される関数名称と正確に一致させておく必要があります。 ファイルにはメイン関数をサポートするためのその他いくつもの UDF が含まれていますが、 少なくとも一つはエクスプレッション内部で参照される名称と引数の個数が一致する UDF を含んでおく必要があります。

UDF 間のデータ交換はそのタイプで制限されます。 UDF から UDF の呼び出しは LScript における呼び出しと全く同じです。

例を使い、 要求されたディレクトリの中に以下のファイルが存在すると仮定しましょう ：

```
¥NewTek¥LScripts¥expressions¥library.ls
¥NewTek¥LScripts¥expressions¥channelValue.ls
```

"**library.ls**" ファイルには以下のコンテンツが含まれています ：

```
locate_channel: fullchannel
{
   parts = parse(".",fullchannel);

   group = ChannelGroup(parts[1]);  // ルートのチャンネルグループからはじめます
   lastgroup = group;
   subgroup = nil;
   x = 2;
```

```
while(group)
{
   // parts[x] と一致するサブグループをスキャン
   // 一致しなかった場合は、 チャンネル名称の
   // 開始場所となります

   subgroup = ChannelGroup(group,subgroup);
   last if !subgroup;

   if(subgroup.name == parts[x])
   {
      group = subgroup;
      lastgroup = group;
      subgroup = nil;
      ++x;
   }
}

if(!lastgroup) return(nil);

// parts[] 配列内部に残されているのはチャンネル名称自身の
// コンポーネントです。 チャンネル検索にこれらを
// 入れ込みましょう

channelname = "";   // 配列の作成は避けます
psize = parts.size();

while(x <= psize)
{
   channelname += parts[x];
   if(x < psize) channelname += ".";
   ++x;
}

// チャンネル名称の検索が可能かどうかを確認するため
// 最終グループにおける定義されたチャンネルをスキャンします

chchannel = lastgroup.firstChannel();
while(chchannel)
{
   last if chchannel.name == channelname;
   chchannel = lastgroup.nextChannel();
}

return(chchannel);
}

// ビルトインの clamp() 関数と置き換えます

clamp: val, lower, upper
{
```

```
                result = val;
    if(val < lower)     result = lower;
    else if(val > upper) result = upper;

    return(result);
}
```

一方で、 ファイル "**channelValue.ls**" には以下のコンテンツが含まれています ：

```
chan;
chanName;

channelValue: channel, time
{
    if(chanName != channel) chan = nil;

    if(!chan)
    {
        // speed 用のチャンネルをキャッシュ
        chan = locate_channel(channel);
        if(!chan) return(0);
        chanName = channel;
    }

    return(chan.value(time));
}
```

レイアウトにおいて、 このようにエクスプレッションを入力することが出来るようになります ：

```
clamp(channelValue("WashLight.Intensity",Time),0.0,1.0)
```

これは **channelValue()** UDF を呼び出しています。 **channelValue()** は実際の LScript Channel Object Agent への文字列チャンネル参照を解決するため locate_channel() UDF( デフォルトのライブラリファイル "**library.ls**" にて定義 ) を呼び出します。 **channel-Value()** UDF は現在時刻における指定した Light Object の強さの値を返します。 この値は ( スクリプトの )**clamp()** UDF ( これもまた "**library.ls**" 内部に定義されています ) へと渡され、 特定の範囲内を維持します。

その代わり、 グラフ編集の直接チャンネル参照構文を UDF の呼び出しで使用することが出来ます ：

```
clamp([WashLight.Intensity,Time],0.0,1.0)
```

エクスプレッションエンジンへと読み込まれている UDF 参照は、 それぞれのファイルが修正された時点から次にエクスプレッションが評価される段階で自動的に更新されます。 例えば、 **channelValue()** を参照し読み込み関数の最終行を変更するようなエクスプレッションでは ：

```
return(chan.value(time) + 1.0);
```

次回エクスプレッションが評価される時点 ( 次のフレームなど ) で即座に新しい値が返っ

てきます。

---

既存の エクスプレッション **log()** 関数は指定された値の自然対数 ( 基数が **e** ) を計算しています。 これは基数 **10** を求める機能に誤解されがちなので、 新しい関数 **log10()** を追加しました。

# LScript v2.5 リリースノート

## 新機能

イメージフィルターの LScript にイメージのピクセルデータを参照および修正するための新しいメカニズムが搭載されました。 イメージフィルタースクリプトの **process()** 関数に引数が 1 つだけ定義されていた場合、 イメージフィルターオブジェクトエージェントが引数として **process()** 関数に渡されます。

```
process: ifo
{
   ...
}
```

オリジナルの **process()** 関数のインターフェースは、 従来のスクリプトと互換性を取るために引き続き LScript v2.5 で機能します。

以下のデータメンバーがイメージフィルターオブジェクトエージェントで公開されています。

width ( 読み取り専用 )
: カレントバッファのピクセル単位の幅

height ( 読み取り専用 )
: カレントバッファのピクセル単位の高さ

frame ( 読み取り専用 )
: メインイメージのカレントフレーム番号

start ( 読み取り専用 )
: メインイメージのレンダリング開始時間

end ( 読み取り専用 )
: メインイメージのレンダリング終了時間

red[]
: イメージの赤ピクセルバッファの浮動小数点形式の値
配列の添え字の形式は [**column**,**row**]

green[]
: イメージの緑ピクセルバッファの浮動小数点形式の値
配列の添え字の形式は [**column**,**row**]

blue[]
: イメージの青ピクセルバッファの浮動小数点形式の値
配列の添え字の形式は [**column**,**row**]

alpha[]
イメージのアルファピクセルバッファの浮動小数点形式の値
配列の添え字の形式は [**column**,**row**]

special[] ( 読み取り専用 )
スペシャルバッファの値
配列の添え字の形式は [**column**,**row**]

luminous[] ( 読み取り専用 )
ルミナンスバッファの値
配列の添え字の形式は [**column**,**row**]

diffuse[] ( 読み取り専用 )
ディフューズバッファの値
配列の添え字の形式は [**column**,**row**]

specular[] ( 読み取り専用 )
スペキュラーバッファの値
配列の添え字の形式は [**column**,**row**]

mirror[] ( 読み取り専用 )
ミラーバッファの値
配列の添え字の形式は [**column**,**row**]

trans[] ( 読み取り専用 )
トランスバッファの値
配列の添え字の形式は [**column**,**row**]

rawred[] ( 読み取り専用 )
未加工の赤バッファの値
配列の添え字の形式は [**column**,**row**]

rawgreen[] ( 読み取り専用 )
未加工の緑バッファの値
配列の添え字の形式は [**column**,row]

rawblue[] ( 読み取り専用 )
未加工の青バッファの値
配列の添え字の形式は [**column**,**row**]

shading[] ( 読み取り専用 )
シェーディングバッファの値
配列の添え字の形式は [**column**,**row**]

shadow[] ( 読み取り専用 )
シャドーバッファの値
配列の添え字の形式は [**column**,**row**]

geometry[] ( 読み取り専用 )
ジオメトリバッファの値

配列の添え字の形式は [**column**,**row**]

depth[] ( 読み取り専用 )
デプスバッファの値
配列の添え字の形式は [**column**,**row**]

diffshade[] ( 読み取り専用 )
ディフューズシェードバッファの値
配列の添え字の形式は [**column**,**row**]

specshade[] ( 読み取り専用 )
スペキュラーシェードバッファの値
配列の添え字の形式は [**column**,**row**]

motionx[] ( 読み取り専用 )
X 方向モーションバッファの値
配列の添え字の形式は [**column**,**row**]

motiony[] ( 読み取り専用 )
Y 方向モーションバッファの値
配列の添え字の形式は [**column**,**row**]

reflectred[] ( 読み取り専用 )
赤リフレクションバッファの値
配列の添え字の形式は [**column**,**row**]

reflectgreen[] ( 読み取り専用 )
緑リフレクションバッファの値
配列の添え字の形式は [**column**,**row**]

reflectblue[] ( 読み取り専用 )
青リフレクションバッファの値
配列の添え字の形式は [**column**,**row**]

以下のメソッドも使用することができます :

copy(**<left>**,**<top>**,**<right>**,**<bottom>**)

指定した領域のイメージデータを使用して新しいイメージバッファをカレントバッファに作成します。 **select()** メソッドで使用するバッファ識別子が返されます。

paste(**<bufferid>**,**<left>**,**<top>**)

指定したバッファのイメージデータを現在選択されているバッファの指定した左上のオフセット位置に貼り付けます。 **bufferid** は **copy()** メソッドが返す値を使用します。

select([**<bufferid>**])

指定したイメージバッファをカレントバッファとして選択します。 すべてのデータメンバとメソッドはこのメソッドの呼び出しによって選択されたバッファに対して機能します。 引数を指定しないで呼び出した場合は、メインバッファがカレントになります。

イメージデータとインターフェースを持つイメージフィルターオブジェクトエージェントを使用することによって、 従来のメソッド （**bufferline()**、 **processrgb()** など） よりも高速な処理を行うことができます。

イメージフィルターオブジェクトエージェントで参照および変更するピクセルデータは、 すべて浮動小数点形式で行われます。

LightWave のイメージフィルター "Black & White" の **process()** 関数をイメージフィルターオブジェクトエージェントを使用した LScript で書き換えると下記のようになります。

```
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;
      }
   }
}
```

以下の例は **copy()**, **paste()**, **select()** メソッドを使用して垂直方向に 2 つのイメージの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);
}
```

---

新しいサーフェイス関連のオブジェクトエージェントが幾つか LScript に追加されました。 1 つめはすべての基本となるサーフェイスオブジェクトエージェントです。 **Surface()** コンストラクターは既存のサーフェイスに対してオブジェクトエージェントを生成したり、 新しいサーフェイスを作成するために使用することがでます。

コンストラクターは引数なしで呼び出すと、 システムに定義されている最初のサーフェイスが返されます。

整数値を伴って呼び出すと、 システム内のインデックス値に対応したサーフェイスが返されます。

サーフェイス名称を引数に指定してコンストラクターを呼び出した場合、 システムに既に該当するサーフェイスが存在していたならばそのサーフェイス名称を返します。 存在しなかった場合は、 レイアウト上では 'nil' の値が返され、 モデラー上ではカレントなオブジェクトに対してその名前のサーフェイスが新しく作成されます。

メッシュオブジェクトエージェントを引数にした場合、 その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が返されます。

レイアウト上では、 最初の引数にサーフェイスを受け取るメッシュオブジェクトエージェント、 二番目の引数にサーフェイス名称を指定することによって、 オブジェクトに対して新しいサーフェイスを作成することができます。

以下のチャンネル定数がサーフェイスオブジェクトエージェント用に LScript で定義されています。

| | | |
|---|---|---|
| SURFCOLR | Base Color | (vector) |
| SURFLUMI | Luminosity | (number) |
| SURFDIFF | Diffuse | (number) |
| SURFSPEC | Specularity | (number) |
| SURFREFL | Reflectivity | (number) |
| SURFTRAN | Transparency | (number) |
| SURFTRNL | Translucency | (number) |
| SURFRIND | IOR | (number) |
| SURFBUMP | Bump | (number) |
| SURFGLOS | Glossiness | (number) |
| SURFBUF1 | Special Buffer 1 | (number) |
| SURFBUF2 | Special Buffer 2 | (number) |
| SURFBUF3 | Special Buffer 3 | (number) |
| SURFBUF4 | Special Buffer 4 | (number) |
| SURFSHRP | Diffuse Sharpness | (number) |
| SURFSMAN | Smoothing Angle | (degrees) |
| SURFRSAN | Reflection Seam Angle | (degrees) |
| SURFTSAN | Refraction Seam Angle | (degrees) |
| SURFRBLR | Reflection Blur | (number) |
| SURFTBLR | Refraction Blur | (number) |
| SURFCLRF | Color Filter | (number) |
| SURFCLRH | Color Highlights | (number) |
| SURFADTR | Additive Transparency | (number) |
| SURFAVAL | Alpha Value | (number) |
| SURFGVAL | Glow Value | (number) |
| SURFLCOL | Line Color | (vector) |
| SURFLSIZ | Line Size | (number) |
| SURFSIDE | Sidedness | (integer) |
| SURFGLOW | Glow | (Boolean) |
| SURFLINE | Render Outlines | (Boolean) |
| SURFRIMG | Reflection Image | (Image Object Agent) |
| SURFTIMG | Refraction Image | (Image Object Agent) |

SURFVCOL　　Vertex Coloring　　(number)

以下のデータメンバーがサーフェイスオブジェクトエージェントで公開されています。

name
　サーフェイスの名称

以下のメソッドを使用することができます。

getValue(<channel>)

　指定したチャンネルの （上記に定義されている） 設定値を返します。

getEnvelope(<channel>)

　指定したチャンネルのエンベロープオブジェクトエージェントを返します。 存在しない場合は 'nil' を返します。

getTexture(<channel>)

　指定したチャンネルに関連付けられているテクスチャオブジェクトエージェント （下記に記述） を返します。 存在しない場合は 'nil' を返します。

---

テクスチャオブジェクトエージェントは （getTexture() メソッドを使用して） サーフェイスオブジェクトエージェントから生成されます。 このオブジェクトエージェントはサーフェイスの指定したチャンネルに関連するテクスチャーへのインターフェースを提供します。 テクスチャはサーフェイスの装飾を表現するために蓄積された 1 つ以上のレイヤーを含む独自の設定値を持っています。

以下のテクスチャ定数がテクスチャ用に LScript で定義されています。

TXTRIMAGE　　イメージテクスチャタイプ
TXTRPROCEDURE　　プロシージャルテクスチャタイプ
TXTRGRADIENT　　グラディエントテクスチャタイプ

以下のメソッドがオブジェクトエージェントに公開されています。

setChannelGroup(<Channel Group Object Agent>)

　オブジェクトエージェントを指定してテクスチャチャンネルグループを設定します。 テクスチャレイヤーのパラメータに作成されたエンベロープはこのグループに所属します。

getChannelGroup()

　テクスチャのチャンネルグループオブジェクトエージェントを返します。

firstLayer()

　テクスチャに定義されているの最初のレイヤーのテクスチャーレイヤーオ

ブジェクトエージェント （下記に記述） を返します。

nextLayer(**<TextureLayer Object Agent>**)

テクスチャの指定したテクスチャレイヤーに続くテクスチャーレイヤーオブジェクトエージェントを返します。

addLayer(**<layer type>**)

**TXTRIMAGE**, **TXTRPROCEDURE**, **TXTRGRADIENT** の 1 つを指定してテクスチャに新しいレイヤーを作成します。 そして、 そのテクスチャーレイヤーオブジェクトエージェントを返します。

---

テクスチャーレイヤーオブジェクトエージェントはテクスチャーオブジェクトエージェントによって作成され、 テクスチャに定義されている各レイヤーの設定値に対するインターフェースを提供します。

以下のチャンネル定数がテクスチャーレイヤーオブジェクトエージェント用に LScript で定義されています。

```
TXLRPOSITION      Position            (vector)
TXLRROTATION      Rotation            (vector)
TXLRSIZE          Size                (vector)
TXLRFALLOFF       Falloff             (vector)
TXLRPROJECT       Projection          (constant)
TXLRAXIS          Texture Axis        (integer)
TXLRWWRAP         Width Wrap          (number)
TXLRHWRAP         Height Wrap         (number)
TXLRCOORD         Coordinate System   (integer: 1==object, 2==world)
TXLRIMAGE         Image               (Image Object Agent)
TXLRVMAP          VMap                (VMap Object Agent)
TXLREFOBJ         Reference Object    (Layout Object Agent)
TXLROPACITY       Opacity             (number)
TXLRANTIALIAS     Antialias           (Boolean)
TXLRAAVALUE       Antialias Threshold (number)
TXLRPIXBLEND      Pixel Blending      (Boolean)
TXLRWREPEAT       Width Repeat        (constant)
TXLRHREPEAT       Height Repeat       (constant)
```

テクスチャの **TXLRPROJECT** 設定は下記の定数として定義されています。

```
TXPJPLANAR
TXPJCYLINDRICAL
TXPJSPHERICAL
TXPJCUBIC
TXPJFRONT
TXPJUVMAP
```

テクスチャの **TXLRWREPEAT** と **TXLRHREPEAT** 設定は下記の定数として定義されています。

TXRPRESET
TXRPREPEAT
TXRPMIRROR
TXRPEDGE

以下のデータメンバーがテクスチャレイヤーオブジェクトエージェントで公開されています。

type
レイヤーのタイプ (**TXTRIMAGE**, **TXTRPROCEDURE**, **TXTRGRADIENT**)

以下のメソッドを使用することができます。

getValue(**<channel>**)

レイヤーの指定したチャンネルの値を返します。

setValue(**<channel>**,**<value>**)

指定したレイヤーチャンネルに値を設定します。

---

以下のレイアウトコマンドがLScriptに統合されています。

MotionBlurDOFPreview()
BoneWeightShade()
BoneXRay()
Redraw()
RedrawNow()
RefreshNow()
MatchGoalOrientation()
KeepGoalWithinReach()
LimitH()
LimitP()
LimitB()
LimitDynamicRange()
MorphMTSE()
MorphSurfaces()
MatteObject()
UnseenByAlphaChannel()
UnaffectedByFog()
ShrinkEdgesWithDistance()
UseMorphedPositions()
FasterBones()
BoneStrengthMultiply()
BoneJointComp()
BoneJointCompParent()
BoneMuscleFlex()
BoneMuscleFlexParent()
ShadowMapFitCone()

ItemVisibility(**1-7**)
ItemColor(**1-15**)

IndirectBounces(**1-8**)
DepthBufferAA(**true/false**)
AlertLevel(**1-3**)
RayRecursionLimit(**0-24**)
RenderThreads(**1**,**2**,**4**,**8**)
MatteColor(**r**,**g**,**b**)
PolygonSize(**<float>**)
PolygonEdgeFlags(**1**,**2**,**4**,**8**,**16**)
PolygonEdgeThickness(**<silhouette>**,**<unshared>**,**<crease>**,**<surface>**,**<other>**)
PolygonEdgeZScale(**0.0-1.0**)
BoneSource(**<Object>**)
BoneMinRange(**0.0-...**)
BoneMaxRange(**0.0-...**)
BoneJointCompAmounts(**<float>**,**<float>**)
BoneMuscleFlexAmounts(**<float>**,**<float>**)
ShadowColor(**r**,**g**,**b**)
ShadowMapFuzziness(<float>)
ShadowMapAngle(**0.5-89.5**)
IncludeObject(**<Object>**)
ExcludeObject(**<Object>**)
RegionPosition(**<left>**,**<top>**,**<width>**,**<height>**)

FogType(**1-4**)
FogMinDistance(**<float>**)
FogMaxDistance(**<float>**)
FogMinAmount(**0.0-1.0**)
FogMaxAmount(**0.0-1.0**)

---

## ビヘイビア ( 振る舞い ) の変更

**ChannelGroup()** コンストラクターはチャンネルグループのトップレベルへのアクセスのみを提供していました。 これは正しいチャンネルを探すためにチャンネルグループの深い階層へ検索することを妨げていました。 その他の引数を指定することによって、**ChannelGroup()** コンストラクターはチャンネルグループの再帰的な処理を可能にし、 既存のチャンネルグループオブジェクトエージェントにも対応するようになります。

以下のコードで解説します。

```
@warnings
@script channel
@version 2.5

ChGroup;

create: channel
{
   // locate the channel group recursively
   ChGroup = nil;
   GetGroupName(channel.id,ChannelGroup());
```

```
    // was it found?
    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;

        GetGroupName(cid,ChannelGroup(chgroup));  // recursive call

        chgroup = chgroup.next();
    }
}
```

---

**min()** 関数と **max()** 関数の引数にベクトルを与えると、 最小値もしくは最大値 （基点からの距離） が返されます。

---

**ctltext()** コントロールのラベルコンポーネントは、 コントロール上の他のテキストよりも高輝度で表示させるために薄い影が付くようになりました。

---

**reqend()** 関数は、対となる **reqbegin()** が呼び出された UDF に対してのみ構文上関連付けられるようになりました。 これはリクエスタが生成された場所以外では、 そのリクエスタに対して **reqend()** を呼び出せないことを意味しています。 他の場所 （たとえば UDF のコールバック関数などから） からリクエスタを終了させたい場合は、 **reqabort()** 関数を使用します。

---

初期化ブロックは、 スクリプトの他の UDF に引数で渡されるようになりました。 ブロックは終端に至るまで配列に変換されます。

---

**BoneFalloffType()** の有効値は **7** から **9** に拡張され、フォールオフタイプの 64 のべき乗と 128 のべき乗にアクセスできるようになりました。

---

VMap オブジェクトエージェントの **setValue()** メソッド （モデラーの LScript でのみ有効） は、 ポイント識別子に **'nil'** の値を受け取れるようになりました。 マッピングされてポイントに対して **'nil'** を指定するとそのポイントの頂点マップが削除されます。

---

requpdate() 関数は、 引数に 1 つ以上のコントロールオブジェクトエージェントの識別子を受け取るようになりました。 指定した識別子が info もしくは listbox であった場合のみアップデートによってリフレッシュが行われます。 この変更により ctlinfo() と ctllistbox() コントロールのアップデートを独立して行うことが可能になります。

---

**setvalue()** は、 **ctltext()** コントロールの内容を変更するために使用されるようになりました。 文字列、 配列の参照、 値の初期化ブロックを指定することができます。

加えて、 データメンバー value を修正することによって、 値を直接的にコントロールに割り当てることも可能です。 コントロールに表示する値を変更するために、 単一の文字列、 文字列の配列、 初期化ブロックを割り当てることができます。

---

**@insert files** や外部オブジェクトエージェントのような外部参照を解決する時、 LScript は LightWave のインストーラが作成する \LightWave\Programs\LScripts ディレクトリを最初にチェックします。 未解決の外部参照のためにスクリプトを選択させるファイルダイアログが表示されたり、 スクリーマネットでのレンダリングが失敗するのを抑止するために、 すべての外部参照モジュールをこの場所に集めてください。

---

チャンネルオブジェクトエージェントのメソッドの **createKey()**, **getKeyValue()**, **setKeyValue()** は、 角度ベースのチャンネルに対して自動的に入力値をラジアンに変換し、 出力値を度 （°） に変換します。

---

オブジェクトエージェントのコンストラクターである **Mesh()**, **Light()**, **Camera()** は、 レイアウト上でオブジェクト ID を引数に指定するようになりました。

---

## バグ修正

**library** コマンドは、 ネィティブな共有ライブラリを使用した時、 正しい関数を使用して参照したディスクのファイルを処理していませんでした。

---

ファイル名称の拡張子 '**.p**' は、 **library** コマンドの正しいネィティブな共有ライブラリタイプとして追加されました。

---

LScript で使用されるオブジェクトのチャンネルグループのエントリーにあるメソッドは必要以上に複雑にであったため、 稀に間違った識別子が使用されていました。

---

LScript の VMap ハンドリングシステムは、 各ポイントに定義されている要素がゼロの VMap （**VMSELECT** など） があることを認識していませんでした。

---

コマンドシーケンスの **EditServer()** 関数を使用した時レイアウトをクラッシュさせていました。

---

前もって値が割り当てられていない文字列のインデックスを使用した一連の配列要素へのアクセスがアプリケーションをクラッシュさせる原因になっていました。 現在は '**nil**' の値が返されます。

---

**ctltext()** コントロールは、**ctlposition()** の呼び出しの幅と高さの設定を参照していました。その結果、 どちらかの値がコントロールの実際の配列よりも小さい場合、 潜在的な表示の問題が起こっていました。

---

**AddCamera()** コマンドは引数のカメラ名称を正しく解釈していませんでした。

---

**getWorldRotation()** 関数が LScript のメソッド起動テーブルから外れていました。その結果参照時に未解決になってしまっていました。

---

**WROTATION** 定数がスクリプト環境変数に定義されていませんでした。

---

**getTag()** と **setTag()** のオブジェクトメソッドが正しく動作していませんでした。 これは修正されました。

---

コンパイルしたスクリプトの **timeout** と **countdown** の機能がプラットフォーム間で機能が違っていました。

---

**ctlfilename()** の引数リストで 4 番目の引数の **save**/**load** フラグだけを指定したい時、３番目の引数の **width** を省略することを許可していませんでした。

---

UDF コールバック関数によって呼び出された UDF の複数の戻り値で作成された配列に割り付けられた文字列がロックされていませんでした。 2 番目の UDF が終了した時に LScript のガベッジコレクタのの餌食になっていました。

---

補助配列の参照が正しく生成されていませんでした。

---

**reqabort()** 関数がノンモーダルのウィンドウで正しく機能していませんでした。

---

イメージフィルターが結合された後でジェネリックスクリプトが有効化された時に発生するコンテキストの衝突がアプリケーションをクラッシュさせていました。

---

**getWorldRotation()** メソッドは、 オブジェクトの回転もしくは親子関係にかかわらず同じ値を返していました。

---

**listbox** と **channel** のリクエスターコントロールは **tab** の下で正しく機能していませんでした。

# LScript v2.4 リリースノート

## 新機能

レイアウトオブジェクトのワールド座標系における回転値が **param()** メソッドの新しいフラグ **WROTATION** の使用、 もしくは **getWorldRotation()** メソッドを使用して計算できるようになりました。 オブジェクトの親子関係が解除された場合は、 **WROTATION**/**getWorldRotation()** は、 **ROTATION**/**getRotation()** と同じ値を返します。

---

**ctlpage()** リクエスタ関数は、 カンマで区切られたリストに追加した Control Object Agents の配列を受け取れるようになりました。 この配列の参照は関数の 2 番目と最後の引数で指定します。

---

**ctlmenu()** という名前の新しいコントロールタイプがリクエスターに追加されました。 このコントロールタイプはボタンを押しているときにポップアップメニューを表示するボタンです。

最初の引数はボタンのタイトルです。 次の引数はメニューに表示するアイテムのリストです。 **ctlpopup()** の場合と同じで、 このリストは配列の参照もしくは初期化された値のブロックになります。 メニューからの選択に対して処理を行うコールバック関数を定義する必要があります。 4 番目の引数はオプションでメニューエントリーを明示的に有効 / 無効にするコールバックを定義します。

メニューのセパレータとして認識される特別なエントリー値があります。 2 つ以上の (=) から始まる入力値は、 ポップアップメニューの指定した位置にセパレータとして追加されます。

コールバック関数に返されるメニューの値は、 オリジナルアイテムのエントリー番号に該当します。 セパレータのエントリーに関しては、 コールバックは呼び出されません。

```
@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);
}
```

---

新しい 2 つのリクエスター関数の **drawtextwidth()** と **drawtextheight()** は、 テキスト文字列のピクセル上の表示幅と高さを決定します。 これらの関数は 1 行の文字列を引数に取り、 表示上のピクセルの幅もしくはピクセルの高さを整数値で返します。

---

インフォーメーション表示コントロールがリクエスターで使えるようになりました。 **ctlinfo()** で作成したコントロールは、 リクエスターパネル上にエリアを定義します。 このエリアには指定した UDF コールバックの呼び出しによって描画処理が行われます。 全ての描画は、 LScript によってこのエリア内にクリッピングされます。

**ctlinfo()** 関数には 3 つの引数があります。 最初の 2 つは表示領域の幅と高さを指定する整数値です。 3 番目の引数はインフォーメーションエリアを再表示するときに呼び出される UDF コールバックを定義します。 この UDF コールバックには引数はありませんが、 リクエスターの全ての描画関数 (**drawtext()**,**drawpixel()** など ) にアクセスできます。

```
@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();
}
```

---

## ビヘイビア ( 振る舞い ) の変更

リクエスター関数 **ctltext()** は、 最初の引数だけもしくは展開されるテキスト文字列を指定する 2 番目の引数で配列の参照を受け取れるように拡張されました。

---

**requpdate()** 関数は、 可視状態にあるすべての **ctlinfo()** エリアの再表示を引き起こします。

---

リクエスター描画関数 **drawborder()** は、 オプションでボーダーラインのへこみ具合を指定する論理値を 5 番目の引数に指定することができるようになりました。 **false** を指定した場合、 ボーダーラインは盛り上がったように表示されます。 **true** を指定した場合は、 くぼんだような表示になります。

---

リクエスター関数 **ctlactive()** と **ctlvisible()** は、 3 番目の引数にコントロール識別子の配列を受け取るように拡張されました。

---

## バグ修正

**points[]** Object Agent のデータメンバーの参照を処理するコードにメモリーリークが存在していました。 このリークは LScript によって確保するメモリーと関連していなかったので、 LScript が終了しても開放されていませんでした。

---

リクエスターキーのハンドリングが確実にアプリケーションをクラッシュさせてしまうバグがありました。

---

LScript のプリプロセッサは、 処理中のスクリプトが実行状態でなく単純にインストール状態であることに気づいていませんでした。 これによりプリプロセッサが不適切なタ

イミングでプラグマ命令を実行してしまっていました。

---

レイアウトスクリプトコンパイラは指定したスクリプトをコンパイルする前にある内部構造のフラグを設定していなかったため、 出力ファイルに主要なアーキテクチャ依存の定数定義が欠落していました。 この定義の欠落はレイアウトの LScript ランタイムシステム上で異常な動作を引き起こす原因になっていました。

# LScript v2.3 リリースノート

## 新機能

Scene Object Agent はレイアウトで動的に更新される設定状態を提供してくれるようになりました。 この情報は **dynamicupdate** データメンバで利用可能であり次の値を一つとります ：

DYNUP_OFF
DYNUP_DELAYED
DYNUP_INTERACTIVE

---

Scene Object Agent でシーン内のオブジェクトの表示状態を決定する新規メソッドが利用出来るようになりました。 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れば、 **visibility()** メソッドは以下のインジケータを一つ返します ：

VIS_VISIBLE
VIS_HIDDEN
VIS_BOUNDINGBOX
VIS_VERTICES
VIS_WIREFRAME
VIS_FFWIREFRAME
VIS_SHADED
VIS_TEXTURED

---

Scene Object Agent にレイアウトのスケマティックビューにおけるオブジェクトエントリの XY 座標値を取得する新規メソッドが追加されました。 オブジェクトの名称もしくはプロキシを提供すると **schemaPosition()** メソッドは位置を表す二つの浮動小数点数値を返します。

CommandSequence 関数 **SchematicPosition()** と組み合わせれば、 レイアウトのスケマティックビューポートの配置をスクリプトで完全に管理できるようになります。

---

Scene Object Agent のデータメンバ **displayopts[]** はレイアウトでの表示設定に関する情報を提供します。 以下に配列内の特定エレメントを記します ：

displayopts[1]　モーションパスが表示されている場合にはブーリアン値 'true'
displayopts[2]　ハンドルが表示されている場合にはブーリアン値 'true'
displayopts[3]　IK チェインが表示されている場合にはブーリアン値 'true'
displayopts[4]　サブパッチ外郭が表示されている場合にはブーリアン値 'true'

displayopts[5]　セーフエリアが表示されている場合にはブーリアン値 'true'
displayopts[6]　フィールドチャートが表示されている場合にはブーリアン値 'true'

---

Scene Object Agent の新規データメンバ **generalopts[]** はレイアウトの一般的な設定に関する情報を提供します。 以下に配列内の特定エレメントを記します ：

generalopts[1]　ツールバー非表示が有効の場合にはブーリアン値 'true'
generalopts[2]　ツールバー位置が右の場合にはブーリアン値 'true'
generalopts[3]　その場でペアレントが有効の場合にはブーリアン値 'true'
generalopts[4]　端数フレームを許可が有効の場合にはブーリアン値 'true'
generalopts[5]　スライダにキー表示が有効の場合にはブーリアン値 'true'
generalopts[6]　実レートで再生が有効の場合にはブーリアン値 'true'

---

Layout Object Agents は新規データメソッド **visibility** をエクスポートし、 ここには Scene Object Agent の **visibility()** メソッドから返される値の一つが含まれています。

---

新規メソッド **schemaPosition()** が Layout Object Agents に追加されました。 このメソッドはレイアウトのスケマティックビューポートにおけるオブジェクト位置を表す二つの浮動小数点数値を返します。

---

Scene Object Agents と Layout Object Agents は新規メソッド **server()** をエクスポートします。 この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。 1 番目の引数は照会されているプラグインクラス、 2 番目の引数は選択している特定のプラグインに対するオプションインデックス値となります。

プラグインクラスは文字定数 ( 例 ： "ItemMotionHandler") でも、 以下に記す定義済み定数を用いても構いません ：

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** はハンドラタイプのサーバー ( 例 : "ItemMotionHandler") を、 **_I** はインターフェースタイプのサーバー ( 例 : "ItemMotionInterface") を示しています。

プラグインのある特定のクラス、 例えば Image Filter クラスや Master クラスなどにはアイテムがありません。 これらのクラスを Layout Object に提供する一方で、 Scene Object Agent で使用する場合には有効な戻り値のみを取得することになります ( 勿論、 アクティブなプラグイン )。 オブジェクトに割り当てられているクラスも同様です。 Scene Object Agent では有効な値を返しません。

---

並べ替えメソッド (**sortA()** と **sortD()**) が配列内における Point と Polygon の ID をサポートするようになりました。

---

新規整数型データタイプメソッド **reduce()** が追加されました。 このメソッドはデータタイプから重複した値を除去してくれます ( 文字列や配列などに便利な機能です )。 重複値は線形順に存在していると予測されます。 つまり他のもう一つの値と隣接しているものと推測されるのです。 **reduce()** を呼び出す前に並べ替えメソッドを用いてください。

以下は **reduce()** メソッドの使用例です。 重複の可能性が高い状況において線形順に並べられているポリゴン ID の配列があるとします。 **sortA()** と **reduce()** を呼び出すことで配列から重複値 ( エントリ ) を取り除いています :

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

---

リクエスタはアイドルタイムコールバックをサポートするオプションをつけました。 **reqpost()**/**reqopen()** 関数は新規引数を二つまで受け付け、 アイドルタイム処理を有効にします。 1 番目の引数には呼び出される UDF の名称を指定します。 UDF は引数を取らずリクエスタパネルへの描画が可能になります。 2 番目のオプション引数はアイドルタイム UDF の呼び出し間で使用されるタイムアウトの間隔 ( ミリ秒単位 ) です。 この値が省略されていれば初期タイムアウト値は 300 ミリ秒 ( 半秒 ) となります。

```
generic
{
   reqbegin("Testing Idle");

   c1 = ctlnumber("Number",1.0);
```

```
    reqpost("idleTest");   // 500 ミリ秒の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;
}
```

ある特定の状況においてはシステムが正確な間隔で UDF を呼び出せない場合もあります。 このため指定したタイムアウト値は近似値に過ぎないのだと考えておいてください。 つまりアイドルタイム UDF が呼び出された時点での経過時間の仮定は出来ないということです。

---

**ctlstring()** や **ctlnumber()** 関数から返されるリクエスタのコントロール ID 値が有効な LScript データタイプとなりました。 これにより新規コントロールデータタイプが自身のオブジェクト属性セットとしてサポート出来るようになりました。 下記はメソッドとデータメンバの初期セットです。 属性セットは近い将来の LScript リリースで拡張される予定です。

新規コントロールデータタイプによりエクスポートされる新規メソッド ：

active([<Boolean>])

このメソッドはコントロールの現在のアクティブ状況を設定します。 オプションのブーリアン値はコントロールをアクティブにするかどうかを決定します。 値が省略されれば暗黙のうちに '**true**' が仮定されます。

visible([<Boolean>])

このメソッドはコントロールの現在の可視状況を設定します。 オプションのブーリアン値はコントロールを可視状態にするかどうかを決定します。 値が省略されれば暗黙のうちに '**true**' が仮定されます。

position(<column>,<row>)

コントロールを開かれているリクエスタパネル上の指定した行 / 列の位置に配置します。 再配置の動作は即座に行われます。 このメソッドを呼び出す前後では必ず **visible()** メソッドを使用してコントロールの移動が正しく処理されているかを確認して下さい。

コントロールデータタイプによりエクスポートされるデータメンバ：

value

このデータメンバはコントロールによって表示されるコンテクスト値を保持しています。 ここから値を読み込んだり書き込むということはコントロール ID に対しそれぞれ **setvalue()** や **getvalue()** 関数を使用するのと同じ処理になります。

active ( 読取専用 )

コントロールの現在のアクティブ状況を示すブーリアン値を返します。 '**true**' であれば現在コントロールはアクティブでありユーザーとのやり取りが可能です。

visible ( 読取専用 )

コントロールの現在の可視状況を示すブーリアン値を返します。 '**true**' であれば現在コントロールはリクエスタパネル上で可視状態にあります。

x ( 読取専用 )

リクエスタパネル上におけるコントロールの現在の X 位置です。

y ( 読取専用 )

リクエスタパネル上におけるコントロールの現在の Y 位置です。

w ( 読取専用 )

コントロールの幅です。

h ( 読取専用 )

コントロールの高さです。

---

リクエスタの描画ツールボックスに新規関数が追加されました。 **drawerase()** 関数は引数に XY 座標値それに幅と高さを指定することでリクエスタパネル上の特定の領域を消去します。

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

---

新規 **reqresize()** 関数を使用してスクリプト内部にリサイズ用コールバック関数を定義している場合、 リクエスタパネルはユーザーによりインタラクティブにリサイズ可能となりました。 この関数には引数が 5 つまで渡されます。 1 番目の引数はスクリプト内部で UDF を識別するための文字列が要求されます。 UDF は二つの整数値の引数、 リク

エスタパネルの新しい幅と高さを示す値を受け取り、 何も返しません。

また幅と高さの最小値 ・ 最大値を表す二組のオプション値も指定できます。 値を省略すると幅と高さの最小値は初期ウィンドウ値、 最大値は画像解像度により制限されます。

以下の Generic のスクリプトはサイズに関わらずコントロールを指定位置にロックすることで新しいリサイズメカニズムの使用方を解説しています ：

```
c1;

generic
{
   reqbegin("Resize Me!");

   c1 = ctlnumber("Number",1.0);

   reqresize("resize",,,640,480);  // 最大幅・高さは 640x480 に設定します

   reqpost();
   reqend();
}

resize: w,h
{
   c1.position(w - c1.w - 5,h - c1.h - 35);
}
```

リクエスタパネルのリサイズは **reqresize()** がコールバック UDF の認識に成功した場合にのみ有効となります。 **reqresize()** を使用していない場合、 パネルは初期サイズにロックされます。

---

LScript は新規プラグインクラスである Custom (Object) をサポートしました。 Item Animation と同様、 Custom Object はその **create()** UDF へのオプション引数としてこのスクリプトが割り当てられているオブジェクトの Object Agent を渡すことが可能です。 また Custom Object は **init()**、 **newtime()** そして **cleanup()** UDF もサポートしています。

Custom Object の **process()** UDF は引数を一つだけ取ります。この引数には Custom Object の機能の根幹となるインターフェイスを提供する Custom Object Agent を指定します。 Custom Object Agent は以下のデータとメソッドをエクスポートします ：

view

どのビューに対して描画を施すのかを示す読取専用のデータメンバです。以下の値のうちの一つです ：

VIEW_ZY
VIEW_XZ
VIEW_XY

VIEW_PERSP
VIEW_LIGHT
VIEW_CAMERA
VIEW_SCHEMA

flags[]

カレントの Custom インスタンスに対する読取専用の設定用配列です。現在はまだ一個のエレメント (**flags[1]**) しか持っていません。 Custom オブジェクトが選択された状態で描画されるときには '**true**' それ以外の場合には '**false**' となります。

setColor(**<r,g,b>|r,g,b[,a]**)

この後の続く描画アクションに対するカラーを設定します。

setPattern(**pat**)

描画パターンを設定します。 デフォルトではソリッドになっていますが以下の値が設定可能です ：

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")

drawPoint(**<pos>[,coord]**)

カレントビュー内の指定したベクトル位置 <pos> に設定済みのカラーとラインパターンを使用してポイントを描画します。 coord パラメーターはオプション ( 初期値は **SYS_Object**) ですが下記の値が指定可能です ：

SYS_WORLD ( もしくは "WORLD" か "GLOBAL")
SYS_OBJECT ( もしくは "OBJECT" か "LOCAL")
SYS_ICON ( もしくは "ICON")

drawLine(**<pos1>,<pos2>[,coord]**)

ベクトル <pos1> からベクトル <pos2> までのラインセグメントを描画します。

drawTriangle(**<pos1>,<pos2>,<pos3>[,coord]**)

ベクトル <pos1>、 <pos2> と <pos3> を頂点に持つ三角形を描画します。

drawCircle(**<pos>,rad[,coord]**)

中心位置 <pos>、 半径 rad の円を描画します。

drawText(**<pos>,text[,coord,[align]]**)

ベクトル位置 <pos> に指定した文字列を描画します。 引数 align には以下の値のうちの一つを指定します ：

LEFT ( もしくは "LEFT")
CENTER ( もしくは "CENTER")
RIGHT ( もしくは "RIGHT")

プラグイン API は計算間において設定情報を維持しませんが、 LScript もまた同様です。 つまり **process()** UDF が呼び出される度に、 カラーはレイアウトインターフェイス内でオブジェクトに設定されているカラーへと戻され、 描画パターンは常に **PATTERN_SOLID** となります。

ここに LightWave SDK において Ernie が提供している "barn" サンプルを複製した簡単な Custom Object LScript を紹介します ：

```
@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]]);
}
```

---

LScript はレイアウトスクリプトとして新規 Object Agent を提供するようになりました ： Particle

この Object Agent はプラグイン API Particle システムサービスの基礎となるインターフェイスを提供します。 グローバルサービス用のドキュメントは LightWave SDK の docs にあります。

Particle Object Agent インスタンスは **Particle()** 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は提供する引数によって新規 Particle データベースを作成したり既存の Particle システムを取得します。

コンストラクタに対し有効な Layout Object Agent が提供されるとオブジェクトに対してアクティブになっている Particle システムのリストが返されます。 有効な

Particle システムが一つもない場合には 'nil' が返されます。

```
...
obj = Mesh("Cow");
part = Particle(obj) || error("No particle systems active!");
...
```

新規 Particle システム作成時には、 パーティクルのタイプを指定しバッファタイプのリストを提供しなくてはいけません。 このバッファは各パーティクルに対し作成され維持されるものです。 パーティクルのタイプは **PART_PARTICLE**( 単一パーティクル ) もしくは **PART_TRAIL**( 尻尾つきのパーティクル ) かになります。 LScript ではおなじみの規定ですが、 ここでも文字列 "**particle**" もしくは "**trail**" を使用して指定することも可能です。

各パーティクルに対して割り当て可能なバッファは以下のとおりです ：

```
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")
```

パーティクルの状態は次の三つのうち、 いずれかになります ： **PART_ALIVE**、**PART_DEAD** または **PART_LIMBO**。 これらの値は **PART_ENABLE** バッファへの読み込み / 書き込み時において使用されます。

Particle Object Agent からは以下のメソッドとデータメンバがエクスポートされます ：

count

インスタンス内で定義されているカレントのパーティクル数 ( 整数値 ) を返します。

save()

保存中にパーティクル特有のデータを書き込みます。

load()

読み込み中にパーティクルデータを読み込みます。

attach(**Object Agent**)

Layout Object Agent に対しパーティクルのインスタンスを割り当てます。 これにより HyperVoxels などのシステムでパーティクルデータへのアクセスが可能になります。

detach(**Object Agent**)

オブジェクトからパーティクルのインスタンスの割り当てを解除します。

reset()

インスタンスから全てのパーティクルをクリアにします。

addParticle()

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**)

インスタンスから指定した index のパーティクルを除去します。

setParticle(**index**,**bufid**,**value**)

index で指定したパーティクルに対しパーティクルバッファ (bufid) 用のデータを設定します。

getParticle(**index**,**bufid**)

index で指定したパーティクル用のパーティクルバッファ (bufid) 内にある値を取得します。

ここにパーティクル SDK の例と同じ動作を行う LScript 版サンプルがあります。 オブジェクトに適用し HyperVoxels 内でオブジェクトを有効にすることでテストが可能です：

```
@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);
}
```

---

以下の CommandSequence 関数がレイアウトの LScript プログラミング環境に追加されました。 LightWave レイアウト 525 のシステムと同期が取れています：

DynamicUpdate(**1-3**)
BoundingBoxThreshold(**threshold**)
AddPosition(**<x,y,z>|x,y,z**)
AddRotation(**<h,p,b>|h,p,b**)
AddScale(**<x,y,z>|x,y,z**)
RadiosityIntensity(**0.0-1.0**)
CausticIntensity(**0.0-1.0**)
ZenithColor(**<r,g,b>|r,g,b**)
SkyColor(**<r,g,b>|r,g,b**)
GroundColor(**<r,g,b>|r,g,b**)
NadirColor(**<r,g,b>|r,g,b**)
EditServer(**class**,**index**)
BoneFalloffType(**1-7**)
ShadowMapSize(**16-32000**)
ShowTargetLines()
MasterPlugins()
Generics()
ClearAudio()
LoadAudio()
PlayAudio()
ReplaceObjectLayer(**layer**,**filename**)
HStiffness(**stiffness**)
PStiffness(**stiffness**)
BStiffness(**stiffness**)
ItemActive(**Boolean**)
ItemLock(**Boolean**)
RadiosityType(**1-3**)
AddButton(**command**,**group**)
LoadObjectLayer(**layer**,**filename**)
LightQuality(**1-5**)

---

新規プラグマ **@name** がプリプロセッサに追加されました。 このプラグマの値はスクリプト用の表示名称となります。 この値は LightWave の LCore サブシステムから使用され、 プラグインとしてインストールされた時点でスクリプト用のメニュー表示名称として提供されます。 さらに将来的には識別関数 **setdesc()** を呼び出す能力を持つスクリプトに対しバックアップ値としても使用されることになります。

```
@warnings
@script generic
@name Bone Cloner
...
```

指定するとコンパイルされた LScript 内にもこのスクリプト名称が埋め込まれ、 インストール時に同一名称がつけられるようになります。

---

VMap Object Agent がモデラーにおいてユーザー定義頂点マップの作成をサポートするようになりました。

新規頂点マップを作成するのも、 定義済みタイプのマップを作成するのもほとんど同じです。 カスタムの頂点マップを作成するため既に定義済みのタイプ ID を使用する代わりに、 カスタムのマップタイプを提供します。 カスタムのマップタイプは一意の 4 文字シーケンスであり初期ブロック内に埋め込まれます。

例えば、 my_normals と名称の 1 ポイントにつき二つのデータエレメントを持つ NMBK タイプの頂点マップを生成するためには ：

```
...
@define VMAP_NMBK  @'N','M','B','K'@
...
normMap = VMap(VMAP_NMBK,"my_normals",2);
```

カスタムの頂点マップはアプリケーションのインターフェイスではどこにも現れないという点に留意してください。 これらはアプリケーションに対し認識されないタイプとなりますので割り当てが行われません。 ただしオブジェクトファイル保存時にファイル内に記憶されオブジェクトファイル内部に書き込まれています。

**VMap()** コンストラクタはモードに関わらず ( カスタムタイプであろうと定義済みタイプであろうと ) 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。

---

## ビヘイビア ( 振る舞い ) の変更

リクエスタパネルは Enter キーを '**OK**' ボタンに、 Escape キーを '**Cancel**' ボタンにマッピングするようになりました ( キーが存在する場合 )。

---

レイアウトの CS コマンド **LoadScene()** と **ClearScene()** は **SCENE** モードで機能する Master スクリプトにおいて使用禁止となりました。 どちらのコマンドを使用してもスクリプトをクリア可能ですが不安定な状態で LScript が残ってしまう可能性があります。

---

Channel Object Agent の **setKeyTime()** メソッドがオブジェクトから削除されました。

キータイムのインデックスは読み込み専用のため、 このメソッドは目的を持たなくなったためです。

---

選択ボタンが押された時点で **ctlfilename()** 関数のデフォルトビヘイビアは保存ダイアログを出すようにしました。 4 番目のオプション引数でブーリアン値 **true** を指定するとこのデフォルトのビヘイビアを変更して読み込みダイアログを開きます。 Mac では保存ダイアログと読み込みダイアログが機能的に決定的に異なるため、 この修正が行われました。

---

**getfile()** に **ctlfilename()** と同様にビヘイビアの修正を行いました。

---

## バグ修正

**AddEnvelope()** と **RemoveEnvelope()** は内部的にエンベロープの名称を"で囲みレイアウトがクラッシュする原因となっていました。

---

**ctlposition()** 関数はモデラースクリプト使用時において正しい引数の個数を受け付けてくれていませんでした。

---

**polygon()** メソッドと Point 整数型データタイプの **polygon** データメンバは正しく実装されていなかったため、 不正な戻り値を返していました。

---

ランタイムシステムはバイトをスワップする必要がある場合、 例えばインテルでコンパイルされたスクリプトをマッキントッシュで実行する場合などに、 **short** 型の整数タイプがうまく管理できていませんでした。 これにより奇妙なエラーメッセージが出てきてクラッシュする原因となっていました。

---

並べ替えメソッド (**sortA()** と **sortD()**) は文字列を適切に扱えていませんでした。 新しい並べ替えのアルゴリズムではこの問題が修正されました。

---

前置と後置インクリメント、 前置と後置デクリメントはオブジェクトのデータメンバに適用した場合に内部的に正しいポインタタイプを提供していませんでした。

---

VMap の **setValue()** メソッドは頂点マップアクセス関数についての私の理解が乏しいた

め、（また SDK における特別な API インターフェイスの整理が乏しいため）バグがありました。 これにより同一 VMap Object Agent を二度目またはサブシーケントで呼び出すとクラッシュしていました。

---

連想代入は割り当てられているデータが複数の Object Agents を返す場合に正しく動作していませんでした。

---

**selpoint()** と **selpolygon()** の **CLEAR** アクションは **ID** または **NDX** フラグを使用し引数として提供している場合に、 カレントの選択から指定ポイントまたはポリゴンをクリアしていませんでした。

---

**ctlimage()** に提供されるイメージオフセット値は正しく処理されていませんでした。

---

**ctlchoice()** 関数へ渡される頂点選択引数が有効になりました。

---

Generic のスクリプトで他の Generic スクリプトを処理の一部として呼び出す CommandSequence 関数を呼び出す場合に再入の問題がありましたが修正されました。

# LScript v2.2 リリースノート

## 新機能

20 個の LScript プラグインが全て単一のディスクファイルに統合されました。 さらに LScript は LCore へと統合され内部サブシステムとなりました。

---

Mesh Object Agent に新規データメンバ "**flags**" が追加されました。 このデータメンバは要素 5 の配列です。 最初の三つの要素には **shadows[]** データメンバと同じ値が入っています。 残り二つにはオブジェクトの現在の " カメラ無効 " と " 光線無効 " 設定を表すフラグ **flags[4]** と **flags[5]** が入っています。

この新規データメンバは既存の **shadows[]** データメンバに取って代わるものです。 近い将来 Object Agent から **shadows[]** データメンバがはずされるときには、 このデータメンバを使用していくことになります。

---

Object Agent クラスに一連の新規識別メソッドが追加されました。 これらのメソッドはオブジェクトタイプを示すブーリアン値 **true/false** を返します。 Object Agent がオブジェクトタイプを代用しています。

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

---

Image Object Agent には二つの新規メソッド **alpha()** と **alphaSpot()** と、 イメージデータのアルファチャンネルにアクセスするための **hasAlpha** データメンバが追加されました。

**hasAlpha** データメンバはイメージがアルファデータを含んでいるか否かを示すブーリアンフラグを返します。 このフラグが **true** であれば新規メソッド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。

新規 **alpha()** メソッドは Image Object Agent の **rgb()** メソッドと同様の動作を行いま

す。 イメージ内部の位置 (x,y) を指定すると、 そのピクセルに対するアルファの値が返されます。 アルファチャンネルデータは各 RGB 一組の値に対し一つの値を持ちます。

**alphaSpot()** は **rgbSpot()** と同様の動作を行いますが、単一の数値 ( アルファ値 ) のみを返します。

---

リクエスタシステムに **reqabort()** 関数が追加されました。 この関数はコントロールやリクエスタのコールバック関数内部から呼び出される関数で、 即座にリクエスタを終了させます。 ユーザーがシステムへの介入を要求されることはありません。

引数無しで **reqabort()** が呼び出されると、 **reqpost()** 関数へは "**Cancel**" の値が返されます。 この場合はユーザーがキャンセルボタンを押す動作を再現します。 **reqabort()** に **true** の値を指定すると、 **reqpost()** には "**OK**" の値が渡されます。

```
generic
{
   reqbegin("Reqabort() Test");

   reqsize(200,180);

   c1 = ctlstate("Goodbye",0,50,"goodbye");

   ctlposition(c1,35,10);

   if(reqpost())
      info("OK を押しました ");
   else
      info("Cancel を押しました ");

   reqend();
}

goodbye: val
{
//    reqabort();
   reqabort(true);
}
```

---

**ctlposition()** 関数には三つの数値オプション引数が追加されました。 この新規引数は LScript でサイズ変更可能なリクエスタコントロールへのアクセスを提供します。

3 番目の引数にはコントロールの幅 ( ピクセル値 ) を、 4 番目の引数はコントロールの高さ ( ピクセル値 ) を指定します。

5 番目の引数にはコントロールのオフセット値を指定します。 オフセット値を使用するとラベルを持つコントロールであればラベルの幅を、 ラベルがないコントロールに対しては左端から右方向へコントロールがシフトする距離を制御出来ます。

---

モデラーの **selpolygon()** 関数に新しい選択タイプが追加されました。 メッシュ内部で **BONE**、 **PATCH** それに **MBALL** エレメントが選択できるようになりました。

**BONE** はスケルゴンと同義語であり、 実際 **SKELEGON** と指定することも可能です。

**MBALL** はメタボール用です。

これらの選択タイプには引数はなく、 カレントメッシュ内部にある指定したタイプのエレメント全てを選択します。

---

Light Object Agent に "**flags**" という新規データメンバが追加されました。 この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。 より厳密にいうと各要素にはライトに関する以下の情報が提供されています。

| | |
|---|---|
| flags[1] | 範囲限定 がオンであれば true |
| flags[2] | 拡散レベル有効 がオンであれば true |
| flags[3] | 反射光有効 がオンであれば true |
| flags[4] | コースティクス有効 がオンであれば true |
| flags[5] | レンズフレア がオンであれば true |
| flags[6] | ヴォリューメトリクス がオンであれば true |

---

Image Filter のアーキテクチャは 5 個の新規バッファタイプをサポートしました。

| | |
|---|---|
| MOTIONX | X 距離値を含む二次元ベクトルに基づいたモーションブラーバッファ |
| MOTIONY | Y 距離値を含む二次元ベクトルに基づいたモーションブラーバッファ |
| REFLECTRED | 鏡面反射の赤のレベル |
| REFLECTGREEN | 鏡面反射の緑のレベル |
| REFLECTBLUE | 鏡面反射の青のレベル |

新規バッファは全て浮動小数点数値であり、 **floatline()** 関数を用いてアクセスするようにして下さい。

---

Scene Object Agent に新規メソッド **getSelect()** が追加されました。 このメソッドは現在シーンで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。 引数を指定せずに呼び出されると、 選択されたアイテム、 メッシュやボーン、 ライトにカメラなど全ての種類が返されます。 フラグの値を引数として指定することで特定の種類だけの選択に限定することが出来ます。 フラグには **MESH**、 **BONE**、 **LIGHT** それに **CAMERA** が指定できます。

このメソッドはどのアイテムが選択されているかを確定する処理を簡易化するために使用されますが、 特別な処理で必要となる **firstSelect()** と **nextSelect()** メソッドもまだ現在でも利用可能です。

---

Scene Object Agent に新規データメンバ **backdroptype** が追加されました。 この値には現在のシーンに設定されている背景の種類が入っています。 値は **SOLID** または **GRADIENT** です。

---

Scene Object Agent に背景関連のメソッドが新たに 2 ～ 3 個追加されました。

| | |
|---|---|
| backdropRay(time,ray) | 指定した時刻において指定した光線が交差する色の値をベクトルとして返します。 光線は正規化されていなければなりません。 |
| backdropColor(time) | 指定された時刻における zenith、sky、ground それに nadir の色の値をベクトルで返します。 |
| backdropSqueeze(time) | 指定した時刻における sky と ground の割合の量を浮動小数点数値で返します。 |

---

Scene Object Agent に新規メソッド **renderCamera()** が追加されました。 メソッドはシーン内で指定した時刻における有効なカメラを Camera Object Agent として返します。

---

Scene Object Agent に新規データメンバ **fogtype** が追加されました。 この値にはシーンでアクティブになっているフォグの種類が書き込まれています。 フォグのタイプは **NONE**、 **LINEAR**、 **NONLINEAR1** もしくは **NONLINEAR2** です。

---

フォグ関連のメソッドが Scene Object Agent にいくつか新規追加されました。

| | |
|---|---|
| fogMinDist(time) | フォグ効果が最小限の時点における視点 ( 一般的にはカメラ ) からの距離を返します。 |
| fogMaxDist(time) | フォグ効果が最大限の時点における距離を返します。 |
| fogMinAmount(time) | 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) | フォグの最大量 ( 最長距離における量 ) を返します。 |
| fogColor(time) | 指定した時刻におけるフォグの色をベクトル形式で返します。 |

---

Scene Object Agent にシーンの現在の合成状況に関する情報を提供するデータメンバが三つ追加されました。 この三つのデータメンバ **compfg**、 **compbg** そして **compfgalpha** にはアクティブな合成画像 ( それぞれ前景、 背景、 前景アルファ ) 用の Image

Object Agent、 もしくはその位置に対して画像がアクティブでない場合には '**nil**' が入っています。

---

新規リクエスタ関数には **requpdate()** が追加されました。 この関数はリクエスタパネル上にあるリストボックスが新しい値で更新できるようにコントロールのコールバック関数の中から呼び出せます。 リストボックスの **count** や **value** コールバック関数内部からは呼び出すことは出来ず、 呼び出したとしても何の影響も与えません。

```
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();
}
```

LScript でバイナリのデータを直接埋め込むことが可能になりました。 このデータは LScript から符号無しバイトのストリームとして扱われます。 これでどんな種類のバイナリデータでもスクリプト内部に埋め込んでおき実行時にアクセス出来るようになります。

バイナリデータは一組のプリプロセッサ宣言子、 "**@data**" と "**@end**" の間に区切られています。 個々のバイトは範囲 0 ～ 255 の整数値として宣言されます。 このデータは各行どんな長さでも可能ですが、 可読性と最大エディタ互換を考慮に入れて 80 列以下に押さえるようにして下さい。

バイナリデータの各ブロックには名前を付けておかなければなりません。 指定された名称は実行開始時においてスクリプトの操作環境内部で大域変数となります。

```
@data bobdata
070 079 082 077 000 000 039 064 076 087 079 050 084 065 071 083 000 000 000
010
069 121 101 000 073 114 105 115 000 000 076 065 089 082 000 000 000 018 000
000
...
@end
```

バイナリブロックはスクリプト内で他の大域変数と同様、 実行時における存在を識別します。 Objects として並べ替えも可能です。

```
info(bobdata.size());
```

繰り返し文 **foreach()** でも使用できます。

```
foreach(x,bobdata)
{
   ...
}
```

また関数へも渡すことが可能です。

```
info(blocksize(bobdata));

blocksize: data
{
   return(data.size());
}
```

これらのデータはスクリプト実行と同時にアクセス ・ 修正が可能です。

```
x = bobdata[25];
bobdata[25] = 0;
```

注釈として付け加えると、 バイナリデータを必要最小限の値にとどめることでスクリプトサイズがかなり小さくなるだけでなく、 バイナリデータの解析が若干速くなります。 例えば "**0**" は "**000**" よりも解析速度が上がり、 ディスクファイル内で二バイト分節約できます。 上記例におけるバイトは見た目に整列できるためだけに 0 が追加されおり、

必ずしも必要ではありません。

ディスクファイルから LScript のバイナリファイルを構文上正しく生成するためには、この Generic スクリプトを使用できます。 これにより新規ファイルへコードを生成したり、 既存ファイル ( 例 スクリプト ) を指定し生成されたコードを付加することも可能になります。

---

バイナリデータのブロックを扱うため特別に新規メソッドが File Object Agent に追加されました。 **writeData()** を使用すれば、 バイナリデータをブロックとしてディスクファイルに送信することが出来ます。 またスクリプト内部に埋め込まれたバイナリデータからバイナリファイルを再作成するためにも使用できます。 例えば LigthWave のオブジェクトファイルをスクリプト内部に埋め込んでおいてデータが必要となるまでスクリプトと共に持たせておくことが可能です。

```
@version 2.2
@warnings

generic
{
   output = File("ball.lwo","wb");
   output.writeData(ball);

   // 重要！ LoadObject() を呼び出す前に
   // 残っているデータをディスクに全てフラッシュして
   // ファイルを閉じておかなくてはなりません

   output.close();

   LoadObject("ball.lwo");

   filedelete("ball.lwo");
}

@data ball
070 079 082 077 000 000 039 064 076 087 079 050 084 065 071 083 000 000 000
010
069 121 101 000 073 114 105 115 000 000 076 065 089 082 000 000 000 018 000
000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 080 078 084
083
...
@end
```

---

LScript の操作環境において新規大域定数値 **SCRIPTID** が追加されました。 この定数には現在実行されているスクリプトのフルパスとファイル名が書き込まれています。

---

リストボックスコントロールにはコントロール内でユーザーから選択されたエントリを

特定出来るオプションのコールバック関数を四つ持つようになりました。 **select** コールバック関数は選択されているエレメントのインデックスを表す整数引数値を受け取り、エレメントが選択可能かどうかを示すブーリアン値 **true/false** を返します。

```
@version 2.2
@warnings

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_select");

    reqpost();
}

lb_count
{
    return(lb_items.size());
}

lb_name: index
{
    return(lb_items[index]);
}

lb_select: index
{
    // 奇数番号のエレメントだけが選択可能

    return(index & 1);
}
```

---

## ビヘイビアの変更

LScript においてリストボックスコントロールは常に複数選択となりました。 これを調整するため、 ある値に変更が加えられました。

リストボックスコントロールの "**event**" 関数へ渡される引数には現在リストボックス内で現在選択されているアイテムを示すインデックスの配列が、 また何も選択されていない場合には '**nil**' が渡されます。

リストボックスコントロール用 **getvalue()** の戻り値タイプは選択されているアイテムのインデックス値を示す複数エレメントとなりました。 **getvalue()** が呼び出されたときにリ

ストボックスでアイテムが選択されていない場合には '**nil**' を返します。

**setvalue()** 関数はリストボックス内での選択設定用に様々なデータタイプの番号を受け付けるようになりました。

ここに新しいリストボックスインターフェイスのメカニズムを解説するスクリプトをご用意しました。

```
@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 = @"Item_2","Item_4"@;
   a = @1,3,5@;
   //   a = 4;
   //   a = "Item_5";

   setvalue(c1,a);
}
```

---

LScript は Panels のサブシステムではなく、 直接 LightWave のツールキットを使用してリクエスタを作成するようになりました。 このためリストボックスの高さの値の解釈に変更が生じています。

リストボックスコントロールは幅をピクセル値で、 高さを可視できるリストボックスのエントリ個数で指定していました。 しかしこれからはリストボックスの高さもピクセル値で指定することになります。 V2.1 リリースノートで提供されているスクリプトの例をとると、 V2.2 では下記のコードは幅 300 ピクセル高さ 10 ピクセルのリストボックスを生成することになってしまいます。

```
c1 = ctllistbox("Items",300,10,"lb_count","lb_name","lb_event");
```

V2.2 で使用するためには更新する必要があります。

```
c1 = ctllistbox("Items",300,300,"lb_count","lb_name","lb_event");
```

---

LScript の実行時システムで使用される圧縮アルゴリズムはコンパイルされたファイルサイズがあるしきい値を超えた場合に正しく機能していませんでした。 このため同様の動作を行いながらも均等な比率で圧縮し、 ファイルサイズに影響を受けることのないバイナリに基づいた新しい圧縮アルゴリズムへと置き換えました。

オリジナルの圧縮アルゴリズムも以前コンパイルされたスクリプトの解凍をサポートするため残されています。

---

Command Sequence 関数の **LightColor()** と **AmbientColor()** は 0.0 は 0、 1.0 は 255 を表す浮動小数点数の形式を引数の型と定めました。 個々のカラー値として範囲 0 ～ 255 の整数値を使用することは可能ですが同等の浮動小数点数値、 範囲 0.0 ～ 1.0 へと変換する必要があります ( 例えば整数値 150 は 150/255.0)。

---

## バグ修正

コンテナ保持配列は新しく値が割り当てられたときに完全に値をクリアにすることなく、 仮想ポインタを再生配列に残したままでした。

---

配列の複写に二次利用を防ぐ追加メモリロックがありました。 このためスクリプト実行中に配列を頻繁に作成 ・ 破棄しているとメモリを浪費していました。

---

モデラーの **new()** コマンドは LightWave [6] のリリースでは整数パラメータを落としていましたので修正しました。

---

オブジェクト処理におけるロジックフローの問題のため特定のデータメンバ ('**totallayers**' など ) が認識されていませんでした。

---

**ctltab()** で作成されたタブコントロールが正しいデータタイプを受け取っていなかったために、 タブが変わるとアプリケーションが落ちる原因となっていました。

---

Item Motion の **create()** 関数へ渡される Object Agent の **genus** データメンバが、オブジェクトのタイプに関わらず常にゼロ (0) を返していました。

---

Image Object Agent の **rgbSpot()**、 **lumaSpot()** と **needAA()** メソッドは引数の個数を正しく受け付けていませんでした。

---

CS の **LightFalloffType()** 関数は文字列配列を不正に受け付けていました。 1(== オフ ) ～ 4 までの整数値を受け付けるように修正しました。

---

V1.4 で紹介されている Point と Polygon データメンバが V2.0 においてオブジェクトコードを再編成していくと失われていました。 これを LScript に追加しなおしました。

---

正規表現を含む変数 ( 検索もしくは検索置換 ) は他の値との一致 ・ 不一致の検査が正しく行われていませんでした。

# LScript v2.1 リリースノート

[ 注記 ]
**プラグイン API が修正されたため、LScript v2.1 を正しく動作させるためには LightWave [6.5] が必要となります。**

## 新機能

レイアウト LScript の **getfirstitem()** コマンドの機能は、 新しい一連の Object Agent コンストラクタにより必要とされなくなりました。 これら新規コンストラクタの出現により **getfirstitem()** 関数は不要となりましたので、 最新版の LScript からは削除されることになるでしょう。

5 つの新規 Objecgt Agent コンストラクタ関数が LScript に追加されました ：

```
Mesh()
Camera()
Image()
Light()
Scene()
```

これらの各 Object Agent コンストラクタは **getfirstitem()** 関数からの返り値と同じ Object Agent が返ってきます。 また各コンストラクタは **getfirstitem()** 関数と同一タイプの引数を受け付けますが、 Object Agent の型は受け付けません ( 例えば **MESH**、**CAMERA**など)。オブジェクトは名称(文字列)やインデックス(整数値)で識別可能です。またそれに加え、 引数なしにコンストラクタを呼び出すことも出来、 システムに最初に発見されるオブジェクト用の Object Agent が生成されます ：

```
obj = Mesh("Cow");       // オブジェクト "Cow" にアクセスします
obj = Camera(2);         // 2 個目のシーンカメラにアクセスします
obj = Light();           // 1 番目のシーンライトにアクセスします
```

**getfirstitem()** から返される Object Agent で、**next()** を用いてそのタイプのオブジェクト全体を通し繰り返すことが可能になります。

これらのコンストラクタと **getfirstitem()** 関数との主な違いは、 **Mesh()** コンストラクタや若干数のメソッド、 また生成された Object Agents によりエキスポートされたデータが、 グローバルであるという点です。 **Mesh()** コンストラクタはモデラーまたはレイアウト、 どちらの LScript からも呼び出すことが出来ます。

モデラーでは、 例外としてカレントのアクティブメッシュにアクセスすることが出来ます。 インデックス値ゼロ (0) を使用すれば、 Object Agent はカレントの選択メッ

シュオブジェクトに対し生成されます。

```
obj = Mesh(0);          // カレントオブジェクトにアクセスします
```

全ての環境 ( 注記の場所以外 ) において **Mesh()** Object Agent から利用できるメソッドは以下のとおりです ：

pointCount([<layer>])

このメソッドは Mesh オブジェクト内、 またはオプションである Mesh の個別レイヤー内にあるポイント総数を返します ( オブジェクト内の有効レイヤーを決めるには新規 '**totallayers**' を使用してください )。

polygonCount([<layer>])

このメソッドは Mesh オブジェクト内、 またはオプションである Mesh の個別レイヤー内にあるポリゴン数を返します。

layer(<point>|<polygon>)

指定されたポイントやポリゴンを含むレイヤー番号を整数値で返します。

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

このメソッドは値を返しますが、 返り値のタイプは二種類あります。 指定される引数によって返り値のタイプは異なります。

引数でポイント ID を指定すれば、 ポイントの位置ベクトルを返します。

引数でポリゴン ID を指定すれば、 ポリゴンのバウンディングボックスの範囲を表す上限 / 下限ベクトルを返します。

引数でレイヤー番号 ID を指定すれば、 レイヤー内にあるメッシュデータのバウンディングボックスの範囲を表す下限 / 上限を返します。

最後に、 引数で何も指定されていない場合には、 全ての有効レイヤー内にあるメッシュデータ全部をあわせたバウンディングボックスの範囲を表す上限 / 下限ベクトルを返します。

vertexCount(<polygon>)

指定されたポリゴンに含まれる頂点総数を、 整数値で返します。

vertex(<polygon>,<index>)

指定されたポリゴンの指定されたインデックスオフセット値の箇所にあるポイント ID を返します。

select()

Agent が代用しているオブジェクトがカレント選択となります。 レイアウトでは、 シーン内でこのオブジェクトを選択します。 モデラーでは、 オブジェクトの編集をアクティブに切り替えます。

select(<point>|<polygon>)　　（モデラー専用）

指定されたポリゴンまたはポイントを選択します。 選択コンポーネントはメッシュ編集に対して一意なコンポーネントですから、 このメソッドはモデラーで実行している場合のみにおいて利用可能です。

Mesh Object Agent 用の新規・修正データメンバは以下のとおりです：

id　　（モデラー専用）

このデータメンバ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にモデラー内部で使用される文字列です。

name　　（モデラー専用）

GUI の中でユーザーに対し表示されるオブジェクト用 ID を表現する文字列です。

filename　　（モデラー専用）

オブジェクトに対するフルパスののファイル名です。 オブジェクトがまだディスクに保存されていない場合には、 この値は 'nil' になります。

---

新規 Object Agent タイプは LScript 内にグローバルに存在し、 LightWave 環境内部において頂点マップにアクセスするためのインターフェイスを提供します。 コンストラクタは **VMap()** と呼ばれ、 大抵の場合はグローバルに使用出来ます。

このコンストラクタは VMap Object Agent を返します。 VMaps は名称 ( 文字列 ) や、 または特定の VMap タイプに対する整数インデックス値による指定が可能です。 VMap タイプは下記のとおりです。

```
VMSELECT        "select"
VMWEIGHT        "weight"
VMSUBPATCH        "subpatch"
VMTEXTURE       "texture"
VMMORPH         "morph"
VMSPOT        "spot"
VMRGB         "rgb"
VMRGBA         "rgba"
```

現在、 システム内に存在している頂点マップがここに用意されていないタイプの頂点マップの場合、 **VMap()** は 'nil' を返します。

特定の頂点マップタイプ引数なしで **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、 他の存在するマップ全てに対し **next()** メソッドを用いて繰り返すことが出来ます。 **next()** を使用して頂点マップを通し反復していくということは、 タイプの境界を重要視しませんので、 特定のタイプだけを処理したいのであれば頂点マップの タイプを見ていく必要があります。

以下のデータメンバは頂点マップ Object Agent によってエキスポートされています：

name

Object Agent に割り当てられている頂点マップの名称。

dimensions

マップ内の各頂点に割り当てられている値の数。 この値はゼロでもかまいません。

type

Object Agent に割り当てられている頂点マップのタイプ。 上記に定義されている定数値 (**VMSELECT**、 **VMWEIGHT** など ) の一つです。

以下のメソッドがエキスポートされます ：

count()

カレント環境内にある Object Agent タイプの頂点マップの総数を返します。

isMapped(<point>)

マップ内に指定されたポイント ID が存在するかしないかを返します。 返り値は **TRUE** または **FALSE** となります。

getValue(<point>[,<index>])

頂点マップ内の指定されたポイント ID に割り当てられている値を返します。 特定のインデックス値が提供されていない場合、 返されたアイテム数は '**dimensions**' データメンバに含まれる値と同値です。 ポイントが頂点マップ内に存在しなかったり、 頂点マップの次元数がゼロの場合には、 '**nil**' が返されます ( このような事態を避けるために **isMapped()** を使用して '**dimensions**' データメンバを確認してください )。

setValue(<point>,<value>|<array>[,<index>]) （ モデラー専用 ）

頂点マップ内の指定するポイント ID の値を設定してください。 個別の値が提供される場合には、 値はポイントに対する最初の値スロットに代入されます。 オプションであるインデックス値はポイントに対し、 値を代入するスロットの位置を指定します。

またポイント ID に対する値として使用するために、 配列値を指定することも可能です。 オプションのインデックスが指定されていない場合には、 頂点マップ内のポイント ID 用の値スロットの、 一致する番号のスロットに指定されたインデックスオフセットの箇所におけるポイント ID 用値スロットから、 配列内のエレメントが代入されていきます。

**setValue()** メソッドを用いる頂点マップの修正は、 メッシュ編集とみなされるため、 モデラー LScript 内でのみ実行が可能です。 このためこのメソッドはレイアウト LScript から生成された Object Agent 内には存在しません。 さらに、 このメソッドの呼び出しを成功させる前に初期化され

た Mesh Edit セッション内部にいなくてはなりません。

以下は既存のウェイト VMap をユーザーが選択できるモデラー LScript の例です。 各値に対して均一なスケーリングファクターを適用します ：

```
@version 2.1
@warnings

main
{
   vmap = VMap(VMWEIGHT) || error("No weight maps in mesh!");

   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 '",vmap-
names[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();
```

```
}
```

---

レイアウト LScript 内部にチャンネルグループを定義するための新しい Object Agent が用意されました。 **ChannelGroup()** Object Agent コンストラクタは ChannelGroup Object Agent を返します。 この Object Agent はインターフェイス内部の標準の Layout Object Agent と似ていますが、 限定されたメソッドやデータメンバのサブセットのみをサポートしています。 主な目的はレイアウト内にあるチャンネルグループを、 " 隠れている " ものも含めて列挙します ( 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もあります )。

レイアウト内にチャンネルグループが何も存在していない場合には、 **ChannelGroup()** は **'nil'** を返します ( 実際にはありそうもない状況ですが、 よいプログラミングを行うためには様々な可能性を考えてコード化しなくてはなりません )。

特定のチャンネルグループ名称を指定せずに **ChannelGroup()** を呼び出すと、 システム内に定義されている最初のチャンネルグループが返されます。 このチャンネルグループから始めて、 **next()** メソッドを使用すれば存在する全てのチャンネルグループを反復することが出来ます。 他のレイアウト Object Agent と同様、 チャンネルグループがこれ以上存在しない地点までくれば、 **'nil'** が返されます。

特定のチャンネルグループ用の Object Agent を作成できるように、 **ChannelGroup()** コンストラクタもまた、 チャンネルグループ ID( を文字列として ) 受け付けます。 例えば、 LW_MasterChannel 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場合などです ：

```
group = ChannelGroup("MC");
```

もちろん、 LW_MasterChannel プラグインがアクティブでない場合には、 **Channel-Group()** を呼び出すと 'nil' が返されます。

以下は ChannelGroup Object Agent からエキスポートされるデータメンバです ：

**name**

Object Agent がプロキシとして提供する場合のチャンネルグループ名称です。

**parent**

カレントの Object Agent の親チャンネルグループです。 親が存在しなければ、 データメンバは **'nil'** となります。

以下はエキスポートされるメソッドです ：

**firstChannel()**
**nextChannel()**

それぞれ Channel Object Agent を返します (Channel Object Agent から利用可能な新規メソッドとデータメンバについては、 このドキュメントで後述されています )。

next()

リスト内の次のチャンネルグループを返します。

---

LScript の内部 IPC メカニズムはカレントホスト ( モデラーやスクリーマーネットなど ) に対し " 局所化 " を行うことが可能です。 これにより同一マシン上の同一 IPC スクリプトを実行するプロセスが、 確立されたキューと抵触しないようにします。 新しい "**@localipc**" プラグマで、 このメカニズムが可能となります。

このメカニズムは MacOS version 9 までは正しく機能しない点に注意してください。 IPC キューはスクリプト内のこのプラグマの存在していようといまいと、 グローバルでありつづけるでしょう。

---

線形配列はオーバーロード代入命令子 '**+=**' を使用して、 追加 ( または作成 ) が可能です。
既存配列に適用するときには、 既存線形配列のエレメントの終端にある新規エレメントの中に追加データが代入されます ：

```
a[1] = "Bob";
a[2] = 1.0;
a += <1,2,3>;       // placed into a[3]
```

それに加え、 オーバーロード '**+=**' 命令子が '**nil**' の値をもつ変数で使用されるとき、新規線形配列は自動的に生成可能です ：

```
vmapnames = nil;
while(vmap)
{
   vmapnames += vmap.name;
   vmap = vmap.next();
}
```

LScript は使用していない変数を自動的に '**nil**' の値で初期化するため、 上記コードで解説している明示的な '**nil**' の値の代入は、 変数が '**nil**' 以外の値を持っている場合、もしくはすでに既存配列をもっている場合にのみ必要となります。

---

LScript プリプロセッサは新しい順次変数指名子をサポートします。 コンストラクタは整数値で省略記号 ('**..**') を囲む形となります。 この整数値は、 生成する値の範囲として提供されます。 下限整数値の前に基本変数名称があり、 宣言された各変数を生成するために使用されます。

例えば、 この順次変数指名子は ：

```
c1..5;
```

プリプロセッサでは次のように拡張されます ：

```
c1,c2,c3,c4,c5;
```

通常、 順次変数指名子はスクリプトコードの中で、 複数変数を宣言しようとしている箇所ならどこでも使用が可能です。 例えば、 次のコードでは ：

```
(lex1..5) = parse(" ",line);
info(lex1..5);
```

これは機能的には以下の文と同等です ：

```
(lex1,lex2,lex3,lex4,lex5) = parse(" ",line);
info(lex1,lex2,lex3,lex4,lex5);
```

---

新しいパネルベースのリクエスタコントロールが多数、 LScript に追加されました。

ctlpercent(**<title>**,**<initial_value>**)

値をパーセンテージとして表示する浮動小数点数ミニスライダフィールド。 **getvalue()** 関数で浮動小数点数値を返します。

ctlangle(**<title>**,**<initial_value>**)

値を角度として表示する浮動小数点数ミニスライダフィールド。 **getvalue()** 関数で浮動小数点数値を返します。

角度値はラジアンとなっている点に注意してください。 度数で動作させたい場合、 度数とラジアンの間を変換させるには LScript の **rad()** と **deg()** 関数を使用してください。

ctlchannel(**<title>**,**<width>**,**<height>**[,**<selected_channel>**])

シーン内の チャンネルを持つ全オブジェクトを表示するツリー構造の領域。 個別にチャンネルを選択することも出来ます。

<width> はピクセル値で、 <height> は可視列数で表記される点に注意してください。

<selected_channel> は有効な Channel Object Agent Agent インスタンスでなくてはならない点に注意してください。

**getvalue()** で Channel Object Agent インスタンスを返します。

ctlbutton(**<label>**,**<width>**,**<action_udf>**)

スクリプト内にある定義済みのユーザー定義関数が反応する " 何か動作を行う " ボタンです。

<width> はピクセル値で表記される点に注意してください。

以下のスクリプトでは **ctlbutton()** コントロールのセットアップと使用法について解説しています ：

```
@version 2.1
@warnings

c1..2;

generic
{
    reqbegin("Testing");

    c1 = ctlbutton("Increment",150,"addcount");
    c2 = ctlinteger("Count",1);

reqpost();
}

addcount
{
    setvalue(c2,getvalue(c2) + 1);
}
```

ctlstate(**<label>**,**<initial_value>**,**<width>**,**<action_udf>**)

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、 定義済みユーザー定義関数 (<action_udf>) をスクリプト内で設定します。 **ctlbutton()** と異なる点は、 UDF はブーリアンコントロールのカレント状況を表す値を取得する点です。 **false(0)** はオフを、 **true(1)** はオンを表します。

<width> はピクセル値で表記される点に注意してください。

以下のスクリプトでは **ctlstate()** コントロールのセットアップと使用法を紹介しています ：

```
@version 2.1
@warnings

generic
{
    reqbegin("State Control");

    c1 = ctlstate("Testing",true,100,"stateCallback");

    reqpost();
}

stateCallback: val
{
    info(val);      // 0 - off, 1 - on
}
```

ctllistbox(<title>,<width>,<height>,<count_udf>,<name_udf>[,<event_udf>])

テキストエントリーの集合体を単一列で表示するリストボックスコントロールです。

<width> はピクセル値で、 <height> は可視列数で表記される点に注意してください。

スクリプト内部には二つのユーザー定義関数が定義されていなければなりません。 一つはリストボックス内のアイテム総数を返す関数 (<count_udf>)、 そしてもう一つはリストボックスのインデックスオフセットが指し示す箇所の文字列値を返す関数 (<name_udf>) です。 オプションの <event_udf> はリストボックス内部でイベントに反応したとき ( 例えばアイテムが選択されたとき ) にいつでも、 コントロールを受信するための関数の定義 / 指定が可能です。

<count_udf> は引数は何も受け取らず、 アイテム数を整数値で返します。

<name_udf> は照会された整数インデックス値を受け取り、 スロット内に代入されている単一文字列値を返します。

オプションの <event_udf> は選択アイテムの整数インデックス値を受け取りますが、 何も返しません。

以下のスクリプトでは **ctllistbox()** コントロールのセットアップと使用法を紹介しています ：

```
@version 2.1
@warnings

c1;

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","lb_event");

    reqpost();
}

lb_count
{
     return(lb_items.size());
}

lb_name: index
```

```
{
    return(lb_items[index]);
}

lb_event: index
{
    info("You selected '",lb_items[index],"'!");
}
```

これに加えて、 以下のスクリプトではリストボックスのコンテンツを管理またはやりとりするために、 どのようにしてボタンコントロールが使用出来るのか紹介しています ：

```
@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
{
    // たとえ 'nil' であっても
    // ここでは全てのエレメント総数をカウントする
    // size() を使用しないでください。

    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());
}
```

パネルサブシステムのリフレッシュ方式のため、 開いているパネルのリフレッシュに対しリストボックスを取得するのに、 リストボックス内の現在の選択を更新しなくてはなりません。 現在、 リストボックス内で選択されているアイテムをもう一度選択した場合、 パネルはリフレッシュを避けるように最適化されてきました。 このためリフレッシュに反応する毎に新しい値を選択するようにしなくてはいけません。

---

Modeler[6] をサポートするために、 数種類の新しい CommandSequence 関数がモデラー LScript に追加されました。

close()

アクティブオブジェクトを閉じます。 オブジェクトが修正されているなら、 除去する前にメッシュデータを保存するよう促されます。

closeall()

モデラー内の全てのオブジェクトを閉じます。 修正されているオブジェクトがあれば、 除去する前にメッシュデータを保存するよう促されます。

exit()

モデラーを終了します。 システム内に修正されているメッシュデータがあれば、 モデラーを終了する前に保存するよう促すプロンプトが表示されます。

swaphidden()

カレントオブジェクト内の隠れたメッシュデータを可視状態にするために、 また現在、 可視状態のメッシュを隠すようにします。

setlayername([<string>])

現在、 選択されているフォアグラウンドレイヤーに対し名称を設定します。 引数を指定せずにこの関数を呼び出すと、 以前に設定されていたレイヤー名称が削除されます。

setobject(<string>[,<index>])

アクティブなモデラーオブジェクトを選択します。 オプションのインデッ

クス値は、 同一名称を持つオブジェクトの中からどのオブジェクトを選択するかを指定します。

setpivot(<vector>)

カレントオブジェクトに対し、 ピボットポイント用の位置を設定します。

unweld()

2 個以上のポリゴンで共有されている頂点をばらばらにします。 各ポリゴンが完全に共有されていないポイント群を持つためです。

weldaverage()

---

**smoothshift()** コマンドは、 第 3 引数としてオプションの引数を持つことになりました。この引数は適用するスケーリングファクターを浮動小数点数値で指定します。

smoothshift(<distance>[,<maxangle>[,<scale>]])

---

新しい **time()** 関数では現在時刻に対する複数アイテムとして、 時、 分、 秒、 チック値を返します。 引数として時刻値が指定されている場合には ( 時刻値は 1900 年 1 月 1 日からの経過秒数として定義されています )、 返り値はタイムインデックスに対する相対値となります。

時は 24 時間ミリタリーフォームですので、 値の範囲は 0 ～ 23 となります。 分と秒の範囲は 0 ～ 59 となります。

'**tick**' の値は 1900 年 1 月 1 日からの経過秒数です。

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

---

新しい **date()** 関数からは複数アイテムとして、 日、 月、 年 (4 桁 )、 曜日、 暦 ( ユリウス暦など )、 現在の月に対応する文字列、 曜日に対応する文字列などが返されます。引数として時刻値が指定されている場合には (1900 年 1 月 1 日からの経過秒秒数 )、返り値はタイムインデックスに対する相対値となります。

曜日は日曜日 == 1 から始まり、 日付は 1 ～ 365 となります。

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

---

モデラー LScript の **selpolygon()** 関数は、 **PART** 選択状態を指定できるようになりまし

た。 これで文字列名称でパラメーター化されているポリゴンが選択されるようになります。

```
...
selmode(USER);
selpolygon(SET,PART,"LeftButtock");
...
```

---

以下のレイアウト CommandSequence 関数が LScript(Generic と Master) に追加されました。 この LScript はビルド番号 474 のレイアウトと同期が取れています ：

PreviousSibling()
NextSibling()
CenterItem()
ShowSafeAreas()
ShowFieldChart()
CacheRadiosity()
CacheCaustics()
CacheShadowMap()
EditPlugins()
FitAll()
FitSelected()
EnableVIPER()
RayTraceShadows()
RayTraceReflection()
RayTraceRefraction()
FogType()
FogMinDistance()
FogMaxDistance()
FogMinAmount()
FogMaxAmount()
LightIntensityTool()
TopView()
BottomView()
BackView()
FrontView()
RightView()
LeftView()
SchematicView()
EnableVolumetricLights()
AddPartigon()
EnhancedAA()
PolygonEdgeColor(<red>,<green>,<blue>|<red,green,blue>)
MaskPosition(<left>,<top>,<width>,<height>)
MaskColor(<red>,<green>,<blue>|<red,green,blue>)
IncludeLight(<id>|<index>)
ExcludeLight(<id>|<index>)
Antialiasing([1-9])
AddEnvelope(<channel>)
RemoveEnvelope(<channel>)
FogColor(<red>,<green>,<blue>|<red,green,blue>)
AutoConfirm(true|false)

MorphTarget(<name>|<type>,<index>|<id>)
SaveSceneCopy(<filename>)
SchematicPosition(<x>,<y>)
RadiosityTolerance(<number>)
SaveObject(<filename>)
SaveObjectCopy(<filename>)
SaveTransformed(<filename>)

カラーの値は浮動小数点数値として指定して下さい。

引数を指定せずに **Antialiasing()** 関数を呼び出すと、 選択カメラに対するアンチエリアシングをオフにします。 値 1 ～ 9 はビルド番号 459 のレイアウトで使用可能なアンチエリアシングの値に対応しています。

注意 ： LScript v2.1 をビルド番号 474 以前の LightWave レイアウトで使用すると不明の CommandSequence 関数であるというエラーメッセージが表示されてしまいますので、 注意してください。

---

**filestat()** 関数が追加されました。 この関数はディスク上のファイルに割り当てられているシステム関連の様々な値を返します。 引数として適切なファイル名称が渡されると、 **filestat()** はファイルのアクセス日、 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 や NTSC ファイルシステムにおいてのみ有効 )、 ファイル所有者のユーザー ID(UNIX でのみ使用可能 )、 ファイルのグループ ID(UNIX においてのみ有効 ) が ( この順番で ) 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、 アクセス日、 作成日時、 更新日時は **time()** 関数や **date()** 関数の引数として使用が可能です。

---

**addcurve()** 関数にはカーブの状態を示すためのオプションとして第 3 引数を用意しました。 これでカーブのコントロール構造を指定する **START** もしくは **END** フラグを指定できるようになりました。 パラメーターが何も指定されていなければ、 カーブに対しては何のコントロールも適用されません。

---

LScript ツールセットに新しい関数が二つ追加されました。 ホストアプリケーションのバージョン番号などについての情報を集める関数です。

hostVersion()

この関数はホストアプリケーションバージョン番号を浮動小数点数値で返します。 返り値にはメジャーバージョン番号とマイナーバージョン番号が含まれており、 例えば 6.1 といった形で返されます。

hostBuild()

この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。 LightWave のビルド番号はリニア状で、 リセットされることは決してありませんので、 バージョン番号よりもアプリケーション機能セットを決定するのにより正確な番号となります。

---

プリプロセッサの **@if/@end** 条件付コードシステムは、 条件として使用可能な二つの値を新しく持つことになりました。 定数 '**host_version**' にはアプリケーションのカレントバージョンを浮動小数点数値として、 定数 '**host_build**' にはホストアプリケーションのカレントビルド番号を整数値として保持しています。

```
...
@if host_build > 410
...
@end
...
```

---

レイアウト用に新しい二つのリクエスターコマンドが追加されました。 この二つの新規コマンドで LScript 内でノンモーダルなリクエスターパネルが可能となりました。

ノンモーダルモードでパネルを開くためには、 **reqpost()** 関数の代わりに **reqopen()** 関数を使用します。 このモードでは、 **options()** 関数の UDF が終了した後でも、 パネルを開いたままで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においてパネルのコントロール変更の処理を行うためには、 コントロールに対しアクティブなリフレッシュコールバック関数を適用しておかなければなりません。 コントロールが修正されると、 このリフレッシュコールバック関数が LScript から呼び出され、 続いてオブジェクトのスクリーン上の属性をリフレッシュするため、 スクリプトの **process()** UDF がレイアウトから呼び出されます。

**reqisopen()** 関数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、 開かれているかどうかを確認するために使用されます。 現在パネルが開かれていれば、 ブーリアンの **TRUE** が返されます。 ノンモーダルなパネルは **reqend()** 関数を呼び出すことで閉じることが出来ます。

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

   c1 = ctlangle("Heading",rad(myHeading));
   c2 = ctlangle("Pitch",rad(myPitch));
   c3 = ctlangle("Bank",rad(myBank));

   ctlrefresh(c1,"heading_refresh");
   ctlrefresh(c2,"pitch_refresh");
   ctlrefresh(c3,"bank_refresh");
```

```
    reqopen();
}
...
```

ノンモーダルなリクエスタパネルはレイアウトの LScript でのみ使用可能です。 ただし、 Generic クラスのスクリプトは除きます。

---

より完全にするために、 Channel Object Agent にはたくさんの新規データメンバとメソッドが追加されました。

以下のデータメンバが追加されました ：

keyCount

チャンネルに割り当てられているエンベロープ内にあるキーの数を返します。

keys[]

エンベロープ内の全てのキーを含むキー ID を保持した線形配列です。 配列長は常に '**keyCount**' となります。

preBehavior

エンベロープにつけられているプリビヘイビアのタイプです。 **CHAN_RESET**、 **CHAN_CONSTANT**、 **CHAN_REPEAT**、 **CHAN_OSCILLATE**、 **CHAN_OFFSET**、 または **CHAN_LINEAR** です。 文字列値 "**reset**"、 "**constant**"、 "**repeat**"、 "**oscillate**"、 "**offset**"、 または "**linear**" も使用出来ます。

他の大部分のデータメンバと異なり、 このデータメンバは読み込み専用ではありません。 このデータメンバに上記のコンスタンツを代入すると、 実際に割り当てられているエンベロープのプリビヘイビアが修正されます。

postBehavior

エンベロープにつけられているポストビヘイビアのタイプです。 その名称を除けば、 このデータメンバは機能的には '**preBehavior**' データメンバと同一です。 エンベロープに対するポストビヘイビア設定を返したり修正したりします。

以下のメソッドが追加されました ：

keyExists(**<time>**)

指定したタイムインデックスの箇所においてキーが存在しなければ '**nil**' を返し、 存在すればキーフレーム ID を返します。

setKeyValue(**<key>**,**<value>**)

指定した値で、 指定したキーフレーム値を設定します。 <value> は常に浮動小数点数値です。

setKeyTime(<key>,<time>)

指定されたキーフレームのタイムインデックスを修正します。

setKeyCurve(<key>,<shape>)

キーフレームの評価するために使用される補間のタイプを設定します。 CHAN_TCB、 CHAN_HERMITE、 CHAN_BEZIER、 CHAN_LINEAR、 CHAN_STEPPED のうちの一つです。 これに加え、 文字列 "TCB"、 "Hermite"、 "Bezier"、 "Linear"、 または "Stepped" でも指定できます。

setKeyHermite(<key>,<parm1>,<parm2>,<parm3>,<parm4>)

このメソッドを使用するとエルミートスプライン用の四つの係数を設定することが出来ます。 使用法がわからなければ、 このメソッドを使用することもないでしょう。

setKeyTension(<key>,<value>)

指定されたキーフレームにおけるテンションを設定します。

setKeyContinuity(<key>,<value>)

指定されたキーフレームにおけるコンティニュイティを設定します。

setKeyBias(<key>,<value>)

指定されたバイアスにおけるテンションを設定します。

getKeyValue(<key>)

指定されたキーフレームに割り当てられている浮動小数点数値の値を返します。

getKeyTime(<key>)

指定され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>)

指定されたキーフレームに対する曲線タイプを返します。 返される定数値のリストは、 setKeyCurve() のエントリを参照してください。

getKeyHermite(<key>)

指定されたキーフレームにおいて現在使用されているエルミートスプラインの四つの係数を返します。 getKeyCurve() で CHAN_HERMITE が返されない場合には、 このメソッドを呼び出すこともないでしょう。

getKeyTension(<key>)

キーフレームに対するカレントのテンション設定を返します。

getKeyContinuity(**<key>**)

キーフレームに対するカレントのコンティニュイティ設定を返します。

getKeyBias(**<key>**)

キーフレームに対するカレントのバイアス設定を返します。

createKey(**<time>**,**<value>**)

指定されたタイムインデックスの地点に、 指定された初期値で新規キーフレームを作成するメソッドです。 キーがうまく作成できれば、 キーフレーム ID が返されますが、 キー作成に失敗すれば '**nil**' が返されます。 このメソッドを使用すると、 即座に '**keyCount**' と '**keys[]**' データメンバの値が更新されます。

deleteKey(**<key>**)

このメソッドを使用すれば、 チャンネルのエンベロープからキーフレームを削除できます。 キーを削除すると、 リアルタイムに '**keyCount**' と '**keys[]**' データメンバの値が更新されますので、 これらのデータメンバに基づいてループ内でこのメソッドを使用している場合には、 慎重に作業する必要があります。

キーフレーム ID はスクリプトから直接使用することは出来ず、 整数型として返されます。 このキーフレーム ID の唯一の目的は、 ID を必要とするメソッドの引数として使用するためだけです。

---

LScript に新しく Envelope Object Agent が追加されました。 **Envelope()** コンストラクタは三つの引数を取得しますが、 第 3 引数はオプションとなります ：

Envelope(**<name>**,**<type>**[,**<group>**])

引数 <name> は新規エンベロープ用の文字列 ID です。

<type> には **CHAN_NUMBER**、 **CHAN_DISTANCE**、 **CHAN_PERCENT** または **CHAN_ANGLE** のどれか一つを指定します。

オプション引数 <group> は新規エンベロープを内包するグループを識別する文字列値です。 指定された <group> が存在しなければ、 LScript はそのグループを作成します。

この Object Agent は Channel Object Agent( 上記参照 ) に追加されたのと同じメソッドとデータメンバを共有します。 それに加え、 以下に記すエンベロープ特有のメソッドも提供されています ：

copy(**<Envelope>**)

指定された Envelope Object Agent 内にある値を、 インスタンスへコピーします。

edit()

ユーザーとやり取りできるように、 エンベロープのコンテンツと設定が表示されます。

save()
load()

この二つの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ストリーム入出力を行うために使用されます。 これらの関数は文脈依存、つまりレイアウトの特定の位相からしか呼び出すことが出来ないようになっています。 例えば **save()** メソッドは **save()** UFD の中からしか呼び出せませんし、 **load()** メソッドは **load()** UDF からしか呼び出すことが出来ません。 適切なコンテキスト範囲外からこれらのメソッドを使用しようとすれば、 とんでもない目にあいます。

persist([<Boolean>])

スクリプトから生成されたエンベロープ ( とグループ ) は、 スクリプトが終了した時点で自動的に破壊されます。 このメソッドを呼び出せばレイアウトの内部チャンネルグループリストをそのままにしておくことが出来ます。 引数なしでこのメソッドを呼び出すと、 エンベロープには持続の有効性を示すフラグがたちます。 ブーリアン値 '**false**' を指定すると、 スクリプトが終了した時点で LScript によって削除されます。

---

新しい **format()** 関数は文字列内のトークンを指定された値で置換することにより、 文字列をフォーマットするのに使用できます。 C 言語の **printf()** 関数と同じような関数です。個別のデータエレメントに対しても使用できますが、 配列に対し使用するのに向いています。

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

トークンはドルサイン記号 (**$**) で表されており、 その後に指定された引数から取得した数値インデックス値が続きます。 インデックス値はシーケンシャルである必要はなく、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。

個々の引数は単一データタイプ ( 変数もしくは定数 ) のリストとなるか、 置換で使用される単一配列値を指定することも可能です。 双方とも指定するということは出来ません。

関数からは全ての置換処理を終えた文字列値が返されます ：

```
...
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));
...
```

---

## 変更箇所

以下のリクエスタコントロール作成関数は LScript から削除されました ：

addcontrol()
addtext()
addhsv()
addrgb()
additems()
addcheckbox()
addfilename()

---

以下のモデラーレイヤー管理関数は LScript から削除されました ：

fglayers()
bglayers()
getempty()
getfull()
getemptyfg()
getemptybg()
swaplayers()
setlayer()
setblayer()

---

LightWave [6] では、 パネルインターフェイスメカニズムはもはやプラグインとして実装されていません。 その代わり LightWave 自身のサブシステムとなりました。 こうすることで、 パネルはいかなる LScript でも利用出来るようになりました。 このため、 **reqbegin()** 関数からはパネルプラグインの存在 / 欠落を示す値が返されることはなくなりました。

---

よりローバストなパネルインターフェイスメカニズムへと移行した結果、 モデラー

LScript の **reqbegin()** に指定可能なオプションのブーリアン値を逆に解釈されるようになりました。 過去の LScript リリースでは、 '**true**' を示すオプションのブーリアン値が指定されていれば、 パネルインターフェイスシステムが ( 利用可能であれば ) 使用され、 スクリプトのリクエスタパネルを構築していました。 '**false**' （または値なし ) ではモデラーの内部パネル作成メカニズムが採用されていました。

これが今では完全に逆になっています。 パネルサブシステムがデフォルトで選択されており、 モデラーの内部リクエスタシステムを選択するには '**true**' のブーリアン値を指定しなくてはなりません。 このため LScript v2.1 でスクリプト実行させるためには、 リクエスタを出すモデラースクリプトは全て変更しなくてはならなくなりました。

---

Displacement Map がサポートしている "**BEFOREBONES**" **flag()** の値が "**PREBONE**" に名称変更されました。

---

LScript v2.0 で導入された Channel Objecct Agent コンストラクタは、 全て接頭辞 "**CHAN_**" を含むように名称が変更されました。

---

## バグ修正

多重エレメント配列 (Init Blocks のような ) を計上する Objecct Agent メソッドでハンドルしている引数を修正しました。

---

Item Animation の **create()** 関数に指定する Object Agent のタイプを、 **MESH** としてコード化していました。

---

再帰的プラグイン呼び出し ( たとえば Item Animation スクリプトに、 そのスクリプトが設定されているオブジェクトのワールド座標値を返すよう指示した場合などに起こります ) は自分自身のインスタンスデータを壊してしまっていました。 というのは内部切り替えスタックが、 インスタンスデータが既にグローバルにインストールされているという状況を認識していなかったためです。

---

配列インデックスとして '**nil**' の値を渡すと、 アプリケーションがクラッシュしていましたので、 修正されました。

---

**lyrsetfg()**、 **lyrsetbg()**、 **lyrswap()** そして **boundingbox()** 関数はモデラーの新しい無制限レイヤーメカニズムに対応していませんでした。 レイヤー数は 32 もしくはそれ以下 ( 整

数ビット数 ) のみに限定されたままでした。

---

File Object Agent の **write()** メソッドは、 引数として線形配列で動作するよう既にコード化されていた一方で、 ファイルがバイナリモードの時には、 ノンバイナリモードで処理を実行 ・ リードしていたため、 メモリがオーバーランしクラッシュしてしまう場合がありました。

---

以下のレイアウト CommandSequence 関数が引数として三つの浮動小数点数値を指定するのと同様に、 ベクトルも指定できるように修正しました ：

Position()
Rotation()
Scale()
PivotPosition()
PivotRotation()

---

Image Object Agent の **rgb()** メソッドは RGB データを返すときに、 間違ったインデックス値を使用していました。

---

Image Object Agent はリクエスタサブシステムから正しく扱われておらず、 データ / メンバにアクセスすることが不可能でした。

---

反復 **foreach()** は変数コンテナの初期化を行わず、 既に保持している値の箇所から反復を始めていました。

---

Object Agent データメンバの妥当性検査が壊れており、 無効なデータメンバ ( 例えば "**shadows**" ではなく "**shadow**") へのリファレンスを拒絶することなく、 許可していました。

---

Mac LScript の **random()** 関数が正しく動作していませんでした。 呼び出すたびに毎回同じ値を返していました。 唯一性を禁じる Apple 哲学によるものだと思われます。

---

リクエスタのセパレーターコントロールは、 **ctlpage()** 関数を使用する特定の Tab コントロールページに割り当てられた時に、 適切な初期化が行われていませんでした。

---

関数引数の変数は渡された値またはデフォルト値を置き換えるために、 UDF 内部に値

を割り当てることが出来ていませんでした。

---

LScript デバッガインターフェイスコードに、 LSIDE デバッガ内にサブ UDF 変数を監視する機能を追加しようとする段階で、 サブ UDF 変数を正しく識別できなくなるバグがありました。

---

CommandSequence を **Master options()** UDF から呼び出すと、 クラッシュしていました。

---

配列の宣言に使用されるインデックス値が 1 より小さくなると、 変なエラーが出てクラッシュしていました。

---

数学代入演算子 **+=**、 **-=**、 ***=**、 それと **/=** は、 左側に整数値が、 右側に浮動小数点数値がくると、 エラーを起こしていました。

# LScript v2.1 リリースノート

**[ 注記 ]**
**プラグイン API が修正されたため、LScript v2.1 を正しく動作させるためには LightWave [6.5] が必要となります。**

## 新機能

レイアウト LScript の **getfirstitem()** コマンドの機能は、 新しい一連の Object Agent コンストラクタにより必要とされなくなりました。 これら新規コンストラクタの出現により **getfirstitem()** 関数は不要となりましたので、 最新版の LScript からは削除されることになるでしょう。

5 つの新規 Objecgt Agent コンストラクタ関数が LScript に追加されました ：

Mesh()
Camera()
Image()
Light()
Scene()

これらの各 Object Agent コンストラクタは **getfirstitem()** 関数からの返り値と同じ Object Agent が返ってきます。 また各コンストラクタは **getfirstitem()** 関数と同一タイプの引数を受け付けますが、 Object Agent の型は受け付けません ( 例えば **MESH**、**CAMERA**など)。オブジェクトは名称(文字列)やインデックス(整数値)で識別可能です。またそれに加え、 引数なしにコンストラクタを呼び出すことも出来、 システムに最初に発見されるオブジェクト用の Object Agent が生成されます ：

```
obj = Mesh("Cow");      // オブジェクト "Cow" にアクセスします
obj = Camera(2);        // 2 個目のシーンカメラにアクセスします
obj = Light();          // 1 番目のシーンライトにアクセスします
```

**getfirstitem()** から返される Object Agent で、**next()** を用いてそのタイプのオブジェクト全体を通し繰り返すことが可能になります。

これらのコンストラクタと **getfirstitem()** 関数との主な違いは、 **Mesh()** コンストラクタや若干数のメソッド、 また生成された Object Agents によりエキスポートされたデータが、 グローバルであるという点です。 **Mesh()** コンストラクタはモデラーまたはレイアウト、 どちらの LScript からも呼び出すことが出来ます。

モデラーでは、 例外としてカレントのアクティブメッシュにアクセスすることが出来ます。 インデックス値ゼロ (0) を使用すれば、 Object Agent はカレントの選択メッシュオブジェクトに対し生成されます。

```
obj = Mesh(0);          // カレントオブジェクトにアクセスします
```

全ての環境 ( 注記の場所以外 ) において **Mesh()** Object Agent から利用できるメソッドは以下のとおりです ：

pointCount([**<layer>**])

このメソッドは Mesh オブジェクト内、 またはオプションである Mesh の個別レイヤー内にあるポイント総数を返します ( オブジェクト内の有効レイヤーを決めるには新規 '**totallayers**' を使用してください )。

polygonCount([**<layer>**])

このメソッドは Mesh オブジェクト内、 またはオプションである Mesh の個別レイヤー内にあるポリゴン数を返します。

layer(**<point>**|**<polygon>**)

指定されたポイントやポリゴンを含むレイヤー番号を整数値で返します。

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

このメソッドは値を返しますが、 返り値のタイプは二種類あります。 指定される引数によって返り値のタイプは異なります。

引数でポイント ID を指定すれば、 ポイントの位置ベクトルを返します。

引数でポリゴン ID を指定すれば、 ポリゴンのバウンディングボックスの範囲を表す上限 / 下限ベクトルを返します。

引数でレイヤー番号 ID を指定すれば、 レイヤー内にあるメッシュデータのバウンディングボックスの範囲を表す下限 / 上限を返します。

最後に、 引数で何も指定されていない場合には、 全ての有効レイヤー内にあるメッシュデータ全部をあわせたバウンディングボックスの範囲を表す上限 / 下限ベクトルを返します。

vertexCount(<polygon>)

指定されたポリゴンに含まれる頂点総数を、 整数値で返します。

vertex(<polygon>,<index>)

指定されたポリゴンの指定されたインデックスオフセット値の箇所にあるポイント ID を返します。

select()

Agent が代用しているオブジェクトがカレント選択となります。 レイアウトでは、 シーン内でこのオブジェクトを選択します。 モデラーでは、 オブジェクトの編集をアクティブに切り替えます。

select(<point>|<polygon>)　　（モデラー専用）

指定されたポリゴンまたはポイントを選択します。 選択コンポーネントはメッシュ編集に対して一意なコンポーネントですから、 このメソッドはモデラーで実行している場合のみにおいて利用可能です。

Mesh Object Agent 用の新規 ・ 修正データメンバは以下のとおりです ：

id　　（モデラー専用）

このデータメンバ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にモデラー内部で使用される文字列です。

name　　（モデラー専用）

GUI の中でユーザーに対し表示されるオブジェクト用 ID を表現する文字列です。

filename　　（モデラー専用）

オブジェクトに対するフルパスののファイル名です。 オブジェクトがまだディスクに保存されていない場合には、 この値は 'nil' になります。

---

新規 Object Agent タイプは LScript 内にグローバルに存在し、 LightWave 環境内部において頂点マップにアクセスするためのインターフェイスを提供します。 コンストラクタは VMap() と呼ばれ、 大抵の場合はグローバルに使用出来ます。

このコンストラクタは VMap Object Agent を返します。 VMaps は名称 ( 文字列 ) や、 または特定の VMap タイプに対する整数インデックス値による指定が可能です。 VMap タイプは下記のとおりです。

```
VMSELECT        "select"
VMWEIGHT        "weight"
VMSUBPATCH        "subpatch"
VMTEXTURE        "texture"
VMMORPH        "morph"
VMSPOT        "spot"
VMRGB        "rgb"
VMRGBA        "rgba"
```

現在、 システム内に存在している頂点マップがここに用意されていないタイプの頂点マップの場合、 **VMap()** は '**nil**' を返します。

特定の頂点マップタイプ引数なしで **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、 他の存在するマップ全てに対し **next()** メソッドを用いて繰り返すことが出来ます。 **next()** を使用して頂点マップを通し反復していくということは、 タイプの境界を重要視しませんので、 特定のタイプだけを処理したいのであれば頂点マップの タイプを見ていく必要があります。

以下のデータメンバは頂点マップ Object Agent によってエキスポートされています ：

name

Object Agent に割り当てられている頂点マップの名称。

dimensions

マップ内の各頂点に割り当てられている値の数。 この値はゼロでもかまいません。

type

Object Agent に割り当てられている頂点マップのタイプ。 上記に定義されている定数値 (**VMSELECT**、 **VMWEIGHT** など ) の一つです。

以下のメソッドがエキスポートされます ：

count()

カレント環境内にある Object Agent タイプの頂点マップの総数を返します。

isMapped(**<point>**)

マップ内に指定されたポイント ID が存在するかしないかを返します。 返り値は **TRUE** または **FALSE** となります。

getValue(**<point>**[,**<index>**])

頂点マップ内の指定されたポイント ID に割り当てられている値を返します。 特定のインデックス値が提供されていない場合、 返されたアイテム数は '**dimensions**' データメンバに含まれる値と同値です。 ポイントが頂点マップ内に存在しなかったり、 頂点マップの次元数がゼロの場合には、 '**nil**' が返されます ( このような事態を避けるために **isMapped()** を使用して '**dimensions**' データメンバを確認してください )。

**setValue(<point>,<value>|<array>[,<index>])** （ モデラー専用 ）

頂点マップ内の指定するポイント ID の値を設定してください。 個別の値が提供される場合には、 値はポイントに対する最初の値スロットに代入されます。 オプションであるインデックス値はポイントに対し、 値を代入するスロットの位置を指定します。

またポイント ID に対する値として使用するために、 配列値を指定することも可能です。 オプションのインデックスが指定されていない場合には、 頂点マップ内のポイント ID 用の値スロットの、 一致する番号のスロットに指定されたインデックスオフセットの箇所におけるポイント ID 用値スロットから、 配列内のエレメントが代入されていきます。

**setValue()** メソッドを用いる頂点マップの修正は、 メッシュ編集とみなされるため、 モデラー LScript 内でのみ実行が可能です。 このためこのメソッドはレイアウト LScript から生成された Object Agent 内には存在しません。 さらに、 このメソッドの呼び出しを成功させる前に初期化された Mesh Edit セッション内部にいなくてはなりません。

以下は既存のウェイト VMap をユーザーが選択できるモデラー LScript の例です。 各値に対して均一なスケーリングファクターを適用します ：

```
@version 2.1
@warnings

main
{
   vmap = VMap(VMWEIGHT) || error("No weight maps in mesh!");

   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 '",vmap-
names[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();
}
```

---

レイアウト LScript 内部にチャンネルグループを定義するための新しい Object Agent が用意されました。 **ChannelGroup()** Object Agent コンストラクタは ChannelGroup Object Agent を返します。 この Object Agent はインターフェイス内部の標準の Layout Object Agent と似ていますが、 限定されたメソッドやデータメンバのサブセットのみをサポートしています。 主な目的はレイアウト内にあるチャンネルグループを、 " 隠れている " ものも含めて列挙します ( 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もあります )。

レイアウト内にチャンネルグループが何も存在していない場合には、 **ChannelGroup()** は **'nil'** を返します ( 実際にはありそうもない状況ですが、 よいプログラミングを行うためには様々な可能性を考えてコード化しなくてはなりません )。

特定のチャンネルグループ名称を指定せずに **ChannelGroup()** を呼び出すと、 システム内に定義されている最初のチャンネルグループが返されます。 このチャンネルグループから始めて、 **next()** メソッドを使用すれば存在する全てのチャンネルグループを反復することが出来ます。 他のレイアウト Object Agent と同様、 チャンネルグループがこれ以上存在しない地点までくれば、 **'nil'** が返されます。

特定のチャンネルグループ用の Object Agent を作成できるように、 **ChannelGroup()** コンストラクタもまた、 チャンネルグループ ID( を文字列として ) 受け付けます。 例えば、 LW_MasterChannel 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場合などです ：

```
group = ChannelGroup("MC");
```

もちろん、 LW_MasterChannel プラグインがアクティブでない場合には、 **ChannelGroup()** を呼び出すと 'nil' が返されます。

以下は ChannelGroup Object Agent からエキスポートされるデータメンバです ：

name

Object Agent がプロキシとして提供する場合のチャンネルグループ名称です。

parent

カレントの Object Agent の親チャンネルグループです。 親が存在しなければ、 データメンバは '**nil**' となります。

以下はエキスポートされるメソッドです ：

firstChannel()
nextChannel()

それぞれ Channel Object Agent を返します (Channel Object Agent から利用可能な新規メソッドとデータメンバについては、 このドキュメントで後述されています )。

next()

リスト内の次のチャンネルグループを返します。

---

LScript の内部 IPC メカニズムはカレントホスト ( モデラーやスクリーマーネットなど ) に対し " 局所化 " を行うことが可能です。 これにより同一マシン上の同一 IPC スクリプトを実行するプロセスが、 確立されたキューと抵触しないようにします。 新しい "**@localipc**" プラグマで、 このメカニズムが可能となります。

このメカニズムは MacOS version 9 までは正しく機能しない点に注意してください。 IPC キューはスクリプト内のこのプラグマの存在していようといまいと、 グローバルでありつづけるでしょう。

---

線形配列はオーバーロード代入命令子 '**+=**' を使用して、 追加 ( または作成 ) が可能です。
既存配列に適用するときには、 既存線形配列のエレメントの終端にある新規エレメントの中に追加データが代入されます ：

```
a[1] = "Bob";
a[2] = 1.0;
a += <1,2,3>;      // placed into a[3]
```

それに加え、 オーバーロード '**+=**' 命令子が '**nil**' の値をもつ変数で使用されるとき、新規線形配列は自動的に生成可能です ：

```
vmapnames = nil;
while(vmap)
{
   vmapnames += vmap.name;
   vmap = vmap.next();
}
```

LScript は使用していない変数を自動的に '**nil**' の値で初期化するため、 上記コードで解説している明示的な '**nil**' の値の代入は、 変数が '**nil**' 以外の値を持っている場合、もしくはすでに既存配列をもっている場合にのみ必要となります。

---

LScript プリプロセッサは新しい順次変数指名子をサポートします。 コンストラクタは整数値で省略記号 ('**..**') を囲む形となります。 この整数値は、 生成する値の範囲として提供されます。 下限整数値の前に基本変数名称があり、 宣言された各変数を生成するために使用されます。

例えば、 この順次変数指名子は ：

```
c1..5;
```

プリプロセッサでは次のように拡張されます ：

```
c1,c2,c3,c4,c5;
```

通常、 順次変数指名子はスクリプトコードの中で、 複数変数を宣言しようとしている箇所ならどこでも使用が可能です。 例えば、 次のコードでは ：

```
(lex1..5) = parse(" ",line);
info(lex1..5);
```

これは機能的には以下の文と同等です ：

```
(lex1,lex2,lex3,lex4,lex5) = parse(" ",line);
info(lex1,lex2,lex3,lex4,lex5);
```

---

新しいパネルベースのリクエスタコントロールが多数、 LScript に追加されました。

ctlpercent(**<title>**,**<initial_value>**)

値をパーセンテージとして表示する浮動小数点数ミニスライダフィールド。 **getvalue()** 関数で浮動小数点数値を返します。

ctlangle(**<title>**,**<initial_value>**)

値を角度として表示する浮動小数点数ミニスライダフィールド。 **getvalue()** 関数で浮動小数点数値を返します。

角度値はラジアンとなっている点に注意してください。 度数で動作させたい場合、 度数とラジアンの間を変換させるには LScript の **rad()** と **deg()** 関数を使用してください。

ctlchannel(**<title>**,**<width>**,**<height>**[,**<selected_channel>**])

シーン内の チャンネルを持つ全オブジェクトを表示するツリー構造の領域。 個別にチャンネルを選択することも出来ます。

<width> はピクセル値で、 <height> は可視列数で表記される点に注意してください。

<selected_channel> は有効な Channel Object Agent Agent インスタンスでなくてはならない点に注意してください。

**getvalue()** で Channel Object Agent インスタンスを返します。

ctlbutton(**<label>**,**<width>**,**<action_udf>**)

スクリプト内にある定義済みのユーザー定義関数が反応する " 何か動作を行う " ボタンです。

<width> はピクセル値で表記される点に注意してください。

以下のスクリプトでは **ctlbutton()** コントロールのセットアップと使用法について解説しています ：

```
@version 2.1
@warnings

c1..2;

generic
{
    reqbegin("Testing");

   c1 = ctlbutton("Increment",150,"addcount");
   c2 = ctlinteger("Count",1);

reqpost();
}

addcount
{
   setvalue(c2,getvalue(c2) + 1);
}
```

ctlstate(**<label>**,**<initial_value>**,**<width>**,**<action_udf>**)

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、 定義済みユーザー定義関数 (<action_udf>) をスクリプト内で設定します。 **ctlbutton()** と異なる点は、 UDF はブーリアンコントロールのカレント状況

を表す値を取得する点です。 **false(0)** はオフを、 **true(1)** はオンを表します。

<width> はピクセル値で表記される点に注意してください。

以下のスクリプトでは **ctlstate()** コントロールのセットアップと使用法を紹介しています ：

```
@version 2.1
@warnings

generic
{
    reqbegin("State Control");

    c1 = ctlstate("Testing",true,100,"stateCallback");

    reqpost();
}

stateCallback: val
{
    info(val);     // 0 - off, 1 - on
}
```

**ctllistbox(<title>,<width>,<height>,<count_udf>,<name_udf>[,<event_udf>])**

テキストエントリーの集合体を単一列で表示するリストボックスコントロールです。

<width> はピクセル値で、 <height> は可視列数で表記される点に注意してください。

スクリプト内部には二つのユーザー定義関数が定義されていなければなりません。 一つはリストボックス内のアイテム総数を返す関数 (<count_udf>)、 そしてもう一つはリストボックスのインデックスオフセットが指し示す箇所の文字列値を返す関数 (<name_udf>) です。 オプションの <event_udf> はリストボックス内部でイベントに反応したとき ( 例えばアイテムが選択されたとき ) にいつでも、 コントロールを受信するための関数の定義 / 指定が可能です。

<count_udf> は引数は何も受け取らず、 アイテム数を整数値で返します。

<name_udf> は照会された整数インデックス値を受け取り、 スロット内に代入されている単一文字列値を返します。

オプションの <event_udf> は選択アイテムの整数インデックス値を受け取りますが、 何も返しません。

以下のスクリプトでは **ctllistbox()** コントロールのセットアップと使用法を紹介しています ：

```
@version 2.1
@warnings

c1;

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","lb_event");

    reqpost();
}

lb_count
{
     return(lb_items.size());
}

lb_name: index
{
    return(lb_items[index]);
}

lb_event: index
{
    info("You selected '",lb_items[index],"'!");
}
```

これに加えて、 以下のスクリプトではリストボックスのコンテンツを管理またはやりとりするために、 どのようにしてボタンコントロールが使用出来るのか紹介しています ：

```
@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
{
    // たとえ 'nil' であっても
    // ここでは全てのエレメント総数をカウントする
    // size() を使用しないでください。

    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());
}
```

パネルサブシステムのリフレッシュ方式のため、 開いているパネルのリフレッシュに対しリストボックスを取得するのに、 リストボックス内の現在の選択を更新しなくてはなりません。 現在、 リストボックス内で選択されているアイテムをもう一度選択した場合、 パネルはリフレッシュを避けるように最適化されてきました。 このためリフレッシュに反応する毎に新しい値を選択するようにしなくてはいけません。

---

Modeler[6] をサポートするために、 数種類の新しい CommandSequence 関数がモデラー LScript に追加されました。

close()

アクティブオブジェクトを閉じます。 オブジェクトが修正されているなら、 除去する前にメッシュデータを保存するよう促されます。

closeall()

モデラー内の全てのオブジェクトを閉じます。 修正されているオブジェクトがあれば、 除去する前にメッシュデータを保存するよう促されます。

exit()

モデラーを終了します。 システム内に修正されているメッシュデータがあれば、 モデラーを終了する前に保存するよう促すプロンプトが表示されます。

swaphidden()

カレントオブジェクト内の隠れたメッシュデータを可視状態にするために、 また現在、 可視状態のメッシュを隠すようにします。

setlayername([<string>])

現在、 選択されているフォアグラウンドレイヤーに対し名称を設定します。 引数を指定せずにこの関数を呼び出すと、 以前に設定されていたレイヤー名称が削除されます。

setobject(<string>[,<index>])

アクティブなモデラーオブジェクトを選択します。 オプションのインデックス値は、 同一名称を持つオブジェクトの中からどのオブジェクトを選択するかを指定します。

setpivot(<vector>)

カレントオブジェクトに対し、 ピボットポイント用の位置を設定します。

unweld()

2 個以上のポリゴンで共有されている頂点をばらばらにします。 各ポリゴンが完全に共有されていないポイント群を持つためです。

weldaverage()

---

**smoothshift()** コマンドは、 第 3 引数としてオプションの引数を持つことになりました。この引数は適用するスケーリングファクターを浮動小数点数値で指定します。

smoothshift(<distance>[,<maxangle>[,<scale>]])

---

新しい **time()** 関数では現在時刻に対する複数アイテムとして、 時、 分、 秒、 チック値を返します。 引数として時刻値が指定されている場合には ( 時刻値は 1900 年 1 月 1 日からの経過秒数として定義されています )、 返り値はタイムインデックスに対する相対値となります。

時は 24 時間ミリタリーフォームですので、 値の範囲は 0 ～ 23 となります。 分と秒の範囲は 0 ～ 59 となります。

'**tick**' の値は 1900 年 1 月 1 日からの経過秒数です。

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

---

新しい **date()** 関数からは複数アイテムとして、 日、 月、 年 (4 桁 )、 曜日、 暦 ( ユリウス暦など )、 現在の月に対応する文字列、 曜日に対応する文字列などが返されます。 引数として時刻値が指定されている場合には (1900 年 1 月 1 日からの経過秒秒数 )、 返り値はタイムインデックスに対する相対値となります。

曜日は日曜日 == 1 から始まり、 日付は 1 〜 365 となります。

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

---

モデラー LScript の **selpolygon()** 関数は、 **PART** 選択状態を指定できるようになりました。 これで文字列名称でパラメーター化されているポリゴンが選択されるようになります。

```
...
selmode(USER);
selpolygon(SET,PART,"LeftButtock");
...
```

---

以下のレイアウト CommandSequence 関数が LScript(Generic と Master) に追加されました。 この LScript はビルド番号 474 のレイアウトと同期が取れています ：

```
PreviousSibling()
NextSibling()
CenterItem()
ShowSafeAreas()
ShowFieldChart()
CacheRadiosity()
CacheCaustics()
CacheShadowMap()
EditPlugins()
FitAll()
FitSelected()
EnableVIPER()
RayTraceShadows()
RayTraceReflection()
RayTraceRefraction()
FogType()
FogMinDistance()
FogMaxDistance()
```

```
FogMinAmount()
FogMaxAmount()
LightIntensityTool()
TopView()
BottomView()
BackView()
FrontView()
RightView()
LeftView()
SchematicView()
EnableVolumetricLights()
AddPartigon()
EnhancedAA()
PolygonEdgeColor(<red>,<green>,<blue>|<red,green,blue>)
MaskPosition(<left>,<top>,<width>,<height>)
MaskColor(<red>,<green>,<blue>|<red,green,blue>)
IncludeLight(<id>|<index>)
ExcludeLight(<id>|<index>)
Antialiasing([1-9])
AddEnvelope(<channel>)
RemoveEnvelope(<channel>)
FogColor(<red>,<green>,<blue>|<red,green,blue>)
AutoConfirm(true|false)
MorphTarget(<name>|<type>,<index>|<id>)
SaveSceneCopy(<filename>)
SchematicPosition(<x>,<y>)
RadiosityTolerance(<number>)
SaveObject(<filename>)
SaveObjectCopy(<filename>)
SaveTransformed(<filename>)
```

カラーの値は浮動小数点数値として指定して下さい。

引数を指定せずに **Antialiasing()** 関数を呼び出すと、 選択カメラに対するアンチエリアシングをオフにします。 値 1 ～ 9 はビルド番号 459 のレイアウトで使用可能なアンチエリアシングの値に対応しています。

注意 ： LScript v2.1 をビルド番号 474 以前の LightWave レイアウトで使用すると不明の CommandSequence 関数であるというエラーメッセージが表示されてしまいますので、 注意してください。

---

**filestat()** 関数が追加されました。 この関数はディスク上のファイルに割り当てられているシステム関連の様々な値を返します。 引数として適切なファイル名称が渡されると、 **filestat()** はファイルのアクセス日、 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 や NTSC ファイルシステムにおいてのみ有効 )、 ファイル所有者のユーザー ID(UNIX でのみ使用可能 )、 ファイルのグループ ID(UNIX においてのみ有効 ) が ( この順番で ) 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、 アクセス日、 作成日時、 更新日時は **time()** 関数や **date()** 関数の引数として使用が可能です。

---

**addcurve()** 関数にはカーブの状態を示すためのオプションとして第 3 引数を用意しました。 これでカーブのコントロール構造を指定する **START** もしくは **END** フラグを指定できるようになりました。 パラメーターが何も指定されていなければ、 カーブに対しては何のコントロールも適用されません。

---

LScript ツールセットに新しい関数が二つ追加されました。 ホストアプリケーションのバージョン番号などについての情報を集める関数です。

hostVersion()

この関数はホストアプリケーションバージョン番号を浮動小数点数値で返します。 返り値にはメジャーバージョン番号とマイナーバージョン番号が含まれており、 例えば 6.1 といった形で返されます。

hostBuild()

この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。 LightWave のビルド番号はリニア状で、 リセットされることは決してありませんので、 バージョン番号よりもアプリケーション機能セットを決定するのにより正確な番号となります。

---

プリプロセッサの **@if/@end** 条件付コードシステムは、 条件として使用可能な二つの値を新しく持つことになりました。 定数 '**host_version**' にはアプリケーションのカレントバージョンを浮動小数点数値として、 定数 '**host_build**' にはホストアプリケーションのカレントビルド番号を整数値として保持しています。

```
...
@if host_build > 410
...
@end
...
```

---

レイアウト用に新しい二つのリクエスターコマンドが追加されました。 この二つの新規コマンドで LScript 内でノンモーダルなリクエスターパネルが可能となりました。

ノンモーダルモードでパネルを開くためには、 **reqpost()** 関数の代わりに **reqopen()** 関数を使用します。 このモードでは、 **options()** 関数の UDF が終了した後でも、 パネルを開いたままで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においてパネルのコントロール変更の処理を行うためには、 コントロールに対しアクティブなリフレッシュコールバック関数を適用しておかなければなりません。 コントロールが修正されると、 このリフレッシュコールバック関数が LScript から呼び出さ

れ、 続いてオブジェクトのスクリーン上の属性をリフレッシュするため、 スクリプトの **process()** UDF がレイアウトから呼び出されます。

**reqisopen()** 関数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、 開かれているかどうかを確認するために使用されます。 現在パネルが開かれていれば、 ブーリアンの **TRUE** が返されます。 ノンモーダルなパネルは **reqend()** 関数を呼び出すことで閉じることが出来ます。

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

   c1 = ctlangle("Heading",rad(myHeading));
   c2 = ctlangle("Pitch",rad(myPitch));
   c3 = ctlangle("Bank",rad(myBank));

   ctlrefresh(c1,"heading_refresh");
   ctlrefresh(c2,"pitch_refresh");
   ctlrefresh(c3,"bank_refresh");

   reqopen();
}
...
```

ノンモーダルなリクエスタパネルはレイアウトの LScript でのみ使用可能です。 ただし、 Generic クラスのスクリプトは除きます。

---

より完全にするために、 Channel Object Agent にはたくさんの新規データメンバとメソッドが追加されました。

以下のデータメンバが追加されました ：

keyCount

チャンネルに割り当てられているエンベロープ内にあるキーの数を返します。

keys[]

エンベロープ内の全てのキーを含むキー ID を保持した線形配列です。 配列長は常に '**keyCount**' となります。

preBehavior

エンベロープにつけられているプリビヘイビアのタイプです。 **CHAN_RESET**、 **CHAN_CONSTANT**、 **CHAN_REPEAT**、 **CHAN_OSCILLATE**、 **CHAN_OFFSET**、 または **CHAN_LINEAR** です。 文字列値 "**reset**"、 "**constant**"、 "**repeat**"、 "**oscillate**"、 "**offset**"、 または

"**linear**" も使用出来ます。

他の大部分のデータメンバと異なり、 このデータメンバは読み込み専用ではありません。 このデータメンバに上記のコンスタンツを代入すると、実際に割り当てられているエンベロープのプリビヘイビアが修正されます。

postBehavior

エンベロープにつけられているポストビヘイビアのタイプです。 その名称を除けば、 このデータメンバは機能的には '**preBehavior**' データメンバと同一です。 エンベロープに対するポストビヘイビア設定を返したり修正したりします。

以下のメソッドが追加されました ：

keyExists(**<time>**)

指定したタイムインデックスの箇所においてキーが存在しなければ '**nil**' を返し、 存在すればキーフレーム ID を返します。

setKeyValue(**<key>**,**<value>**)

指定した値で、 指定したキーフレーム値を設定します。 <value> は常に浮動小数点数値です。

setKeyTime(**<key>**,**<time>**)

指定されたキーフレームのタイムインデックスを修正します。

setKeyCurve(**<key>**,**<shape>**)

キーフレームの評価するために使用される補間のタイプを設定します。 **CHAN_TCB**、 **CHAN_HERMITE**、 **CHAN_BEZIER**、 **CHAN_LINEAR**、 **CHAN_STEPPED** のうちの一つです。 これに加え、 文字列 "**TCB**"、 "**Hermite**"、 "**Bezier**"、 "**Linear**"、 または "**Stepped**" でも指定できます。

setKeyHermite(**<key>**,**<parm1>**,**<parm2>**,**<parm3>**,**<parm4>**)

このメソッドを使用するとエルミートスプライン用の四つの係数を設定することが出来ます。 使用法がわからなければ、 このメソッドを使用することもないでしょう。

setKeyTension(**<key>**,**<value>**)

指定されたキーフレームにおけるテンションを設定します。

setKeyContinuity(**<key>**,**<value>**)

指定されたキーフレームにおけるコンティニュイティを設定します。

setKeyBias(**<key>**,**<value>**)

指定されたバイアスにおけるテンションを設定します。

getKeyValue(**<key>**)

指定されたキーフレームに割り当てられている浮動小数点数値の値を返します。

getKeyTime(**<key>**)

指定され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>**)

指定されたキーフレームに対する曲線タイプを返します。 返される定数値のリストは、 **setKeyCurve()** のエントリを参照してください。

getKeyHermite(**<key>**)

指定されたキーフレームにおいて現在使用されているエルミートスプラインの四つの係数を返します。 **getKeyCurve()** で **CHAN_HERMITE** が返されない場合には、 このメソッドを呼び出すこともないでしょう。

getKeyTension(**<key>**)

キーフレームに対するカレントのテンション設定を返します。

getKeyContinuity(**<key>**)

キーフレームに対するカレントのコンティニュイティ設定を返します。

getKeyBias(**<key>**)

キーフレームに対するカレントのバイアス設定を返します。

createKey(**<time>**,**<value>**)

指定されたタイムインデックスの地点に、 指定された初期値で新規キーフレームを作成するメソッドです。 キーがうまく作成できれば、 キーフレーム ID が返されますが、 キー作成に失敗すれば '**nil**' が返されます。 このメソッドを使用すると、 即座に '**keyCount**' と '**keys[]**' データメンバの値が更新されます。

deleteKey(**<key>**)

このメソッドを使用すれば、 チャンネルのエンベロープからキーフレームを削除できます。 キーを削除すると、 リアルタイムに '**keyCount**' と '**keys[]**' データメンバの値が更新されますので、 これらのデータメンバに基づいてループ内でこのメソッドを使用している場合には、 慎重に作業する必要があります。

キーフレーム ID はスクリプトから直接使用することは出来ず、 整数型として返されます。 このキーフレーム ID の唯一の目的は、 ID を必要とするメソッドの引数として使用するためだけです。

LScript に新しく Envelope Object Agent が追加されました。 **Envelope()** コンストラクタは三つの引数を取得しますが、 第 3 引数はオプションとなります ：

Envelope(**<name>**,**<type>**[,**<group>**])

引数 <name> は新規エンベロープ用の文字列 ID です。

<type> には **CHAN_NUMBER**、 **CHAN_DISTANCE**、 **CHAN_PERCENT** または **CHAN_ANGLE** のどれか一つを指定します。

オプション引数 <group> は新規エンベロープを内包するグループを識別する文字列値です。 指定された <group> が存在しなければ、 LScript はそのグループを作成します。

この Object Agent は Channel Object Agent( 上記参照 ) に追加されたのと同じメソッドとデータメンバを共有します。 それに加え、 以下に記すエンベロープ特有のメソッドも提供されています ：

copy(**<Envelope>**)

指定された Envelope Object Agent 内にある値を、 インスタンスへコピーします。

edit()

ユーザーとやり取りできるように、 エンベロープのコンテンツと設定が表示されます。

save()
load()

この二つの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ストリーム入出力を行うために使用されます。 これらの関数は文脈依存、つまりレイアウトの特定の位相からしか呼び出すことが出来ないようになっています。 例えば **save()** メソッドは **save()** UFD の中からしか呼び出せませんし、 **load()** メソッドは **load()** UDF からしか呼び出すことが出来ません。 適切なコンテキスト範囲外からこれらのメソッドを使用しようとすれば、 とんでもない目にあいます。

persist([**<Boolean>**])

スクリプトから生成されたエンベロープ ( とグループ ) は、 スクリプトが終了した時点で自動的に破壊されます。 このメソッドを呼び出せばレイアウトの内部チャンネルグループリストをそのままにしておくことが出来ます。 引数なしでこのメソッドを呼び出すと、 エンベロープには持続の有効性を示すフラグがたちます。 ブーリアン値 '**false**' を指定すると、 スクリプトが終了した時点で LScript によって削除されます。

新しい **format()** 関数は文字列内のトークンを指定された値で置換することにより、 文字列をフォーマットするのに使用できます。 C 言語の **printf()** 関数と同じような関数です。 個別のデータエレメントに対しても使用できますが、 配列に対し使用するのに向いています。

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

トークンはドルサイン記号 (**$**) で表されており、 その後に指定された引数から取得した数値インデックス値が続きます。 インデックス値はシーケンシャルである必要はなく、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。

個々の引数は単一データタイプ ( 変数もしくは定数 ) のリストとなるか、 置換で使用される単一配列値を指定することも可能です。 双方とも指定するということは出来ません。

関数からは全ての置換処理を終えた文字列値が返されます ：

```
...
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));
...
```

---

## 変更箇所

以下のリクエスタコントロール作成関数は LScript から削除されました ：

addcontrol()
addtext()
addhsv()
addrgb()
additems()
addcheckbox()
addfilename()

---

以下のモデラーレイヤー管理関数は LScript から削除されました ：

fglayers()
bglayers()
getempty()
getfull()
getemptyfg()
getemptybg()
swaplayers()
setlayer()
setblayer()

---

LightWave [6] では、 パネルインターフェイスメカニズムはもはやプラグインとして実装されていません。 その代わり LightWave 自身のサブシステムとなりました。 こうすることで、 パネルはいかなる LScript でも利用出来るようになりました。 このため、 **reqbegin()** 関数からはパネルプラグインの存在 / 欠落を示す値が返されることはなくなりました。

---

よりローバストなパネルインターフェイスメカニズムへと移行した結果、 モデラー LScript の **reqbegin()** に指定可能なオプションのブーリアン値を逆に解釈されるようになりました。 過去の LScript リリースでは、 '**true**' を示すオプションのブーリアン値が指定されていれば、 パネルインターフェイスシステムが ( 利用可能であれば ) 使用され、 スクリプトのリクエスタパネルを構築していました。 '**false**' （または値なし ) ではモデラーの内部パネル作成メカニズムが採用されていました。

これが今では完全に逆になっています。 パネルサブシステムがデフォルトで選択されており、 モデラーの内部リクエスタシステムを選択するには '**true**' のブーリアン値を指定しなくてはなりません。 このため LScript v2.1 でスクリプト実行させるためには、 リクエスタを出すモデラースクリプトは全て変更しなくてはならなくなりました。

---

Displacement Map がサポートしている "**BEFOREBONES**" **flag()** の値が "**PREBONE**" に名称変更されました。

---

LScript v2.0 で導入された Channel Objecct Agent コンストラクタは、 全て接頭辞 "**CHAN_**" を含むように名称が変更されました。

---

## バグ修正

多重エレメント配列 (Init Blocks のような ) を計上する Objecct Agent メソッドでハンドルしている引数を修正しました。

---

Item Animation の **create()** 関数に指定する Object Agent のタイプを、 **MESH** としてコード化していました。

---

再帰的プラグイン呼び出し ( たとえば Item Animation スクリプトに、 そのスクリプトが設定されているオブジェクトのワールド座標値を返すよう指示した場合などに起こります ) は自分自身のインスタンスデータを壊してしまっていました。 というのは内部切り替えスタックが、 インスタンスデータが既にグローバルにインストールされているという状況を認識していなかったためです。

---

配列インデックスとして '**nil**' の値を渡すと、 アプリケーションがクラッシュしていましたので、 修正されました。

---

**lyrsetfg()**、 **lyrsetbg()**、 **lyrswap()** そして **boundingbox()** 関数はモデラーの新しい無制限レイヤーメカニズムに対応していませんでした。 レイヤー数は 32 もしくはそれ以下 ( 整数ビット数 ) のみに限定されたままでした。

---

File Object Agent の **write()** メソッドは、 引数として線形配列で動作するよう既にコード化されていた一方で、 ファイルがバイナリモードの時には、 ノンバイナリモードで処理を実行 ・ リードしていたため、 メモリがオーバーランしクラッシュしてしまう場合がありました。

---

以下のレイアウト CommandSequence 関数が引数として三つの浮動小数点数値を指定するのと同様に、 ベクトルも指定できるように修正しました ：

Position()
Rotation()
Scale()
PivotPosition()
PivotRotation()

---

Image Object Agent の **rgb()** メソッドは RGB データを返すときに、 間違ったインデックス値を使用していました。

---

Image Object Agent はリクエスタサブシステムから正しく扱われておらず、 データ / メンバにアクセスすることが不可能でした。

---

反復 **foreach()** は変数コンテナの初期化を行わず、 既に保持している値の箇所から反復を始めていました。

---

Object Agent データメンバの妥当性検査が壊れており、 無効なデータメンバ ( 例えば "**shadows**" ではなく "**shadow**") へのリファレンスを拒絶することなく、 許可していました。

---

Mac LScript の **random()** 関数が正しく動作していませんでした。 呼び出すたびに毎回同じ値を返していました。 唯一性を禁じる Apple 哲学によるものだと思われます。

---

リクエスタのセパレーターコントロールは、 **ctlpage()** 関数を使用する特定の Tab コントロールページに割り当てられた時に、 適切な初期化が行われていませんでした。

---

関数引数の変数は渡された値またはデフォルト値を置き換えるために、 UDF 内部に値を割り当てることが出来ていませんでした。

---

LScript デバッガインターフェイスコードに、 LSIDE デバッガ内にサブ UDF 変数を監視する機能を追加しようとする段階で、 サブ UDF 変数を正しく識別できなくなるバグがありました。

---

CommandSequence を **Master options()** UDF から呼び出すと、 クラッシュしていました。

---

配列の宣言に使用されるインデックス値が 1 より小さくなると、 変なエラーが出てクラッシュしていました。

---

数学代入演算子 **+=**、 **-=**、 ***=**、 それと **/=** は、 左側に整数値が、 右側に浮動小数点数値がくると、 エラーを起こしていました。