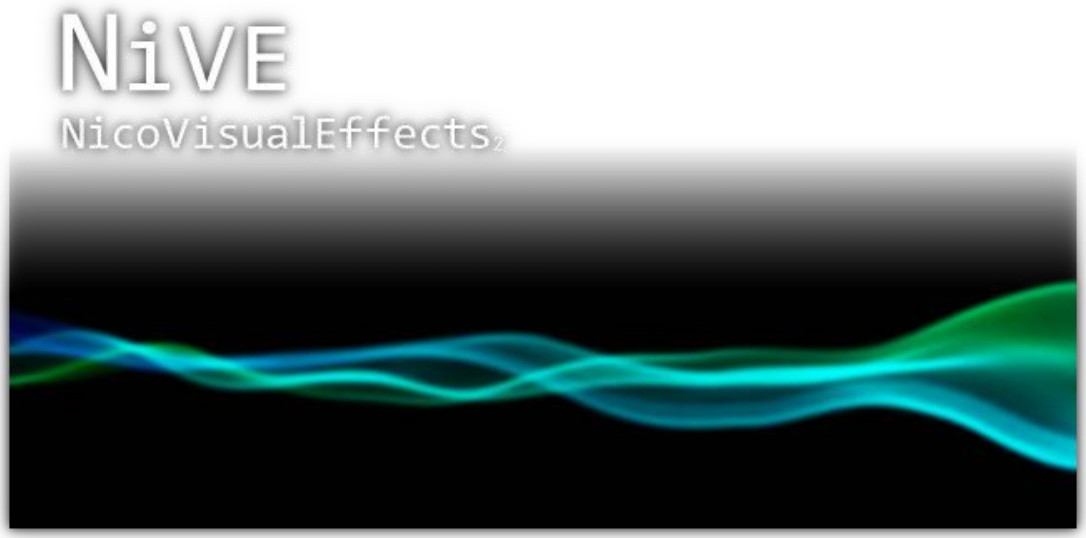

NicoVisualEffects2

コードの記述について

●目次

●はじめに

本テキストでは、プロパティスクリプト、エクスプレッション、オートメーションについてを説明します。これらの機能は、NiVE を C#やスクリプト等、プログラムから操作する機能です。

プロパティスクリプトとは、NiVE で定義された独自の言語を用いてコードの記述することにより、プロパティを操作する機能です。エクスプレッションに比べ、対応しているプロパティや出来ることが少ない代わりに、簡単かつコードの記述量を少なくすることが出来ます。

エクスプレッションは、エフェクトやマテリアルのプロパティをキーフレームによらずに自分で変化させる機能です。この機能は強力で、キーフレーム間の単純な増加や減少といった変化のみならず、Sin 関数を用いた正弦運動や、ある条件が重なったときだけプロパティを変化させるといったことができます。

オートメーションは、レイヤーやコンポジションなどに対し、ある操作を行う機能です。一度に複数のレイヤー、コンポジション、エフェクトを操作できるので、レイヤーのリネームや、特定のエフェクトの削除等、一定の繰り返し動作を一度に行うことが出来ます。

エクスプレッションやオートメーションでは、言語に C#を用いているため、.NET のライブラリをフルに使えます。しかし、これらを使いこなすには計算式や条件を自分で考えることや、C#に関する知識、さらに、レイヤーやエフェクトの各プロパティへの内部的な知識が必要となってきます。このため、初心者はあまり使うべきでない機能ともいえるでしょう。

●プロパティスクリプト

■ 出来ること

プロパティスクリプトでは、一部のプロパティをスクリプトで操作することが出来ます。プロパティスクリプトは、数行程度の簡単な処理を対象としており、複雑な操作をするのには向いていません。
複雑な操作や、非対応のプロパティを操作する場合は、エクスプレッションを使用してください。

■ プロパティスクリプトの仕組み

プロパティスクリプトは、プロパティをすべて取得した後に評価されます。評価順序は不定で、すべてのプロパティスクリプトが評価された後に評価結果が適用されます。

■ エクスプレッションとプロパティスクリプトの評価順序

エクスプレッションとプロパティスクリプトの評価順序は、プロパティスクリプトの評価の後、エクスプレッションが評価されます。

■ 制限

プロパティスクリプトでは、現在評価を行っているプロパティ以外を変更することは出来ず、取得のみ可能です。また、レイヤー、エフェクトのプロパティスクリプトと、入力プロパティのプロパティスクリプトは、それぞれ全く別のタイミングで評価されるため、入力プロパティからレイヤー、エフェクトプロパティ、またはその逆のプロパティを取得することは出来ません。
プロパティスクリプトで扱うことの出来る型は、実数型(double)、ベクトル型(Vertex)、文字列型(string)、真偽値型(bool)のみです。

■ リファレンス

◆ 予約語

プロパティスクリプトでは、以下のキーワードは予約されているため、変数名として使用することは出来ません。

function, break, case, catch, continue, default, do, else, false, finally, for, foreach, goto, if, new, null, return, throw, true, try

◆ 演算子

プロパティスクリプトでは、以下の演算子を使用することが出来ます。各演算子は、括弧の中に書かれている型以外では使用できません。使用されていた場合は Runtime Error が発生します。これらの演算子は代入演算子(=)と組み合わせて使用することが出来ます。同一ではない型同士の演算では、文字列と文字列以外では文字列、ベクトルと実数ではベクトルに変換されます。

+: 加算(実数、ベクトル、文字列)
-: 減算(実数、ベクトル)
*: 乗算(実数、ベクトル)
/: 除算(実数、ベクトル)
%: 剰余(実数、ベクトル)
&: 論理積(真偽値)
|: 論理和(真偽値)

以下の演算子は代入演算子と組み合わせて使用することは出来ません。

!: 否定(真偽値)
&&: 条件 AND(真偽値)
||: 条件 OR(真偽値)

◆ 変数の定義・代入

変数の定義には var キーワードを使用します。変数には、どの方の値でも代入することが出来ます。カンマ(,)による変数の複数定義には対応していません。また、右辺に代入式を記述することは出来ません。メソッド名と同一の変数を定義した場合、そのメソッドは右辺以外に記述できません。

```
var a;
var b = 10;
a = “text”;
a = true;
b = a;
a = null;
a = [800];
b = [10, 20, 7];
a[1] = b[2];
//var c, d; //Compile Error
//a = b = false; //Compile Error
var seedRandom = 10;
//seedRandom(20); //Compile Error
```

◆ 返値

プロパティスクリプトで変更された値を返すには、return キーワードを使用します。プロパティスクリプトからは必ず値を返す必要があります。返値の型は自動的に変換されます。

```
var a = 10;
return a;
```

◆ 比較

プロパティスクリプトでは、以下の演算子を用いることにより、値の比較を行うことが出来ます。比較により得られる値は真偽値となります。各演算子は、括弧の中に書かれている型以外では使用できません。

<: 小なり(実数)
>: 大なり(実数)
<=: 以下(実数)
>=: 以上(実数)
==: 等値(実数、文字列、真偽値)
!=: 非等値(実数、文字列、真偽値)

◆ メソッドの呼び出し

メソッドの呼び出しは、一般的な C スタイル言語と同じように呼び出すことが出来ます。

```
var len = strLen(“Hello World!”);
timeToTimecode(time - getLayerInfo(“inPoint”));
```

◆ ブロック(スコープ)

プロパティスクリプトでは、一般的な言語と同じように中括弧{}でコードを囲むことにより、変数の有効範囲を変更することが出来ます。

```
var a = 0;
{
    //var a = “a”; //Compile Error
    var b = 1;
    a += b;
}
//a += b; //Compile Error
var b = false;
a -= b; //Runtime Error
```

◆ 条件分岐

条件分岐を行うには、if キーワード、else キーワードを使用します。判定式は真偽値で行われますが、真偽値以外の場合は強制的に真偽値に変換されます。変換できない場合、Runtime Error が発生します。

```
if (getLayerInfo(“turn”) == 1)
{
    return 10;
}
else if (-1) //true
{
    if (“true”) //true
        return true;
    else
        return false;
}
```

◆ エラー処理

プロパティスクリプト実行中にエラー(Runtime Error)が発生すると、エラーが発生したプロパティスクリプトは停止し、コードが修正されるまで評価されなくなります。エラーが発生した場合に処理を変更する場合、try キーワードと catch キーワードを使用します。また、catch キーワードの後に新しい変数名を括弧の中に記述することにより、エラーメッセージを取得することが出来ます。

```
var a = null;
try
    return pow(a, 2); //Runtime Error
catch
{
    try
    {
        return a + 10; //Runtime Error
    }
    catch (e)
        return e;
}
```

◆ コメント

コード内にコメントを挿入するには、//、または/**/を使用します。

```
var a = 10; //行末までコメント
/*
  複数行コメント
*/return a;
```

NicoVusualEffects$_2$
コードの記述について

■ ライブラリ

以下にプロパティスクリプトで使用できる定数、メソッドを記述します。引数や返値に型が記述されていますが、プロパティスクリプトのコンパイル時には考慮されず、実行時に評価されます。型が違う場合、Runtime Error が発生します。

定数

◆ double PI
円周率 π を表します。

◆ double E
自然対数の底を表します。

◆ double SQRT1_2
1/2 の平方根を表します。

◆ double SQRT2
2 の平方根を表します。

◆ double LN10
$\log_e 10$ を表します。

◆ double LN2
$\log_e 2$ を表します。

◆ double LOG10E
$\log_{10} e$ を表します。

◆ double LOG2E
$\log_2 e$ を表します。

◆ double time
プロパティスクリプトが評価されているときのコンポジション時間、またはレイヤー時間を表します。

◆ object thisProperty
対象のプロパティの元の値を表します。プロパティによって型が変化します。

メソッド

◆ double abs(double value)
Vertex abs(Vertex value)
指定された値の絶対値を返します。
引数:
value: 絶対値を求める値。
返値: value の絶対値。

◆ double sin(double a)
Vertex sin(Vertex a)
指定された角度のサインを返します。
引数:
a: 弧度法による角度。
返値: a のサイン。

◆ double cos(double a)
Vertex cos(Vertex a)
指定された角度のコサインを返します。
引数:
a: 弧度法による角度。
返値: a のコサイン。

◆ double tan(double a)
Vertex tan(Vertex a)
指定された角度のタンジェントを返します。
引数:
a: 弧度法による角度。
返値: a のタンジェント。

◆ double asin(double x)
Vertex asin(Vertex x)
指定された値のアークサインを返します。
引数:
x: 弧度法による角度。
返値: x のアークサイン。

◆ double acos(double x)
Vertex acos(Vertex x)
指定された値のアークコサインを返します。
引数:
x: 弧度法による角度。
返値: x のアークコサイン。

◆ double atan(double x)
Vertex atan(Vertex x)
指定された値のアークタンジェントを返します。
引数:
x: 弧度法による角度。
返値: x のアークタンジェント。

◆ double atan2(double y, double x)
Vertex atan2(Vertex y, double x)
Vertex atan2(double y, Vertex x)
Vertex atan2(Vertex y, Vertex x)
y/x のアークタンジェントを返します。
引数:
y: y 座標の値。
x: x 座標の値。
返値: y/x のアークタンジェント。

◆ double sqrt(double d)
Vertex sqrt(Vertex d)
指定された値の平方根を求めます。
引数:
d: 平方根を求める値。
返値: d の平方根。

- double exp(double d)
  Vertex exp(Vertex d)
  e の累乗を求めます。
  引数:
  d: 累乗する値。
  返値: e の d 乗した値。

- double pow(double x, double y)
  Vertex pow(Vertex x, double y)
  Vertex pow(double x, Vertex y)
  Vertex pow(Vertex x, Vertex y)
  指定した値を、指定した値で累乗します。
  引数:
  x: 累乗される値。
  y: 累乗する値。
  返値: x の y 乗した値。

- double log(double d)
  Vertex log(Vertex d)
  double log(double a, double newBase)
  Vertex log(Vertex a, double newBase)
  Vertex log(double a, Vertex newBase)
  Vertex log(Vertex a, Vertex newBase)
  自然対数の底、または指定した底での対数を求めます。
  引数:
  d, a: 対数を求める値。
  newBase: 対数の底。
  返値: 求めた対数。

- double log10(double d)
  Vertex log10(Vertex d)
  10 を底とする対数を求めます。
  引数:
  d: 対数を求める値。
  返値: 求めた対数。

◆ double ceil(double a)
Vertex ceil(Vertex a)
指定した値以上の最小の整数値を求めます。
引数:
a: 求める値。
返値: 求めた整数値。

◆ double floor(double a)
Vertex floor(Vertex a)
指定した値以下の最大の整数値を求めます。
引数:
a: 求める値。
返値: 求めた整数値。

◆ double min(double a, double b)
Vertex min(Vertex a, double b)
Vertex min(double a, Vertex b)
Vertex min(Vertex a, Vertex b)
指定した 2 つの値のうち、小さい方の値を返します。
引数:
a: 比較する 1 つめの値。
b: 比較する 2 つめの値。
返値: 2 つのうちの小さい方の値。

◆ double max(double a, double b)
Vertex max(Vertex a, double b)
Vertex max(double a, Vertex b)
Vertex max(Vertex a, Vertex b)
指定した 2 つの値のうち、大きい方の値を返します。
引数:
a: 比較する 1 つめの値。
b: 比較する 2 つめの値。
返値: 2 つのうちの大きい方の値。

◆ double sign(double value)
Vertex sign(Vertex value)
指定した値の符号を返します。
引数:
value: 符号を求める値。
返値: 求めた符号。

◆ double round(double a)
Vertex round(Vertex a)
double round(double value, double digit)
Vertex round(Vertex value, double digit)
Vertex round(double value, Vertex digit)
Vertex round(Vertex value, Vertex digit)
指定した値を最も近い整数値、または指定した小数部の桁数に丸めます。
引数:
a, value: 丸める値。
digit: 小数部の桁数。この値の小数点以下は切り捨てられます。
返値: 丸められた値。

◆ Vertex dot(Vertex a, Vertex b)
指定したベクトル同士の内積を計算します。
引数:
a: 内積を計算する1つめのベクトル。
b: 内積を計算する2つめのベクトル。
返値: 指定したベクトルの内積。

◆ Vertex cross(Vertex a, Vertex b)
指定したベクトル同士の外積を計算します。
引数:
a: 外積を計算する1つめのベクトル。
b: 外積を計算する2つめのベクトル。
返値: 指定したベクトルの外積。

◆ double length(Vertex v)
double length(Vertex point1, Vertex point2)
ベクトル、または2点間の距離を取得します。
引数:
v: 長さを計算するベクトル。
point1: 始点となる頂点。
point2: 終点となる頂点。
返値: ベクトル、または2点間の距離。

◆ Vertex normalize(Vertex v)
ベクトルを正規化します。
引数:
v: 正規化するベクトル。
返値: 正規化したベクトル。

◆ double clamp(double value, double limit1, double limit2)
Vertex clamp(Vertex value, double limit1, double limit2)
Vertex clamp(double value, Vertex limit1, double limit2)
Vertex clamp(double value, double limit1, Vertex limit2)
Vertex clamp(Vertex value, Vertex limit1, double limit2)
Vertex clamp(Vertex value, double limit1, Vertex limit2)
Vertex clamp(double value, Vertex limit1, Vertex limit2)
Vertex clamp(Vertex value, Vertex limit1, Vertex limit2)
値を2つの指定した値の範囲に制限します。
引数:
value: 制限する値。
limit1: 1つめの制限値。
limit2: 2つめの制限値。
返値: 制限値の範囲内であればvalue、そうでなければどちらかの制限値。

◆ double step(double edge, double x)
Vertex step(Vertex edge, double x)
Vertex step(double edge, Vertex x)
Vertex step(Vertex edge, Vertex x)
値が指定した値以上であるかどうかを比較します。
引数:
edge: 基準となる値。
x: 比較対象の値。
返値: edgeがxよりも大きい場合は1.0、そうでない場合は0.0を返します。

◆ bool isNaN(double d)
指定した値がNaN(非数値)であるかどうかを検査します。
引数:
d: 検査する値。
返値: NaNである場合はtrue、そうでない場合はfalse。

◆ bool isInfinity(double d)
指定した値が正の無限大、または負の無限大であるかどうかを検査します。
引数:
d: 検査する値。
返値: 無限大である場合はtrue、そうでない場合はfalse。

◆ bool isPositiveInfinity(double d)
指定した値が正の無限大であるかどうかを検査します。
引数:
d: 検査する値。
返値: 正の無限大である場合はtrue、そうでない場合はfalse。

◆ bool isNegativeInfinity(double d)
指定した値が負の無限大であるかどうかを検査します。
引数:
d: 検査する値。
返値: 負の無限大である場合はtrue、そうでない場合はfalse。

- double strLen(string s)
  文字列の文字数を取得します。
  引数:
  s: 文字数を取得する文字列。
  返値: 取得した文字数。

- bool contains(string s, string value)
  元となる文字列に指定した文字列が含まれているかどうかを検査します。
  引数:
  s: 元となる文字列。
  value: 検索する文字列。
  返値: 指定した文字列が含まれる場合はtrue、そうでない場合はfalse。

- double indexOf(string s, string value)
  元となる文字列から、指定した文字列を検索し、最初に見つかった位置を返します。
  引数:
  s: 元となる文字列。
  value: 検索する文字列。
  返値: 指定した文字列が見つかった場合は0から始まるインデックス、見つからなかった場合は-1。

- double lastIndexOf(string s, string value)
  元となる文字列から、指定した文字列を検索し、最後に見つかった位置を返します。
  引数:
  s: 元となる文字列。
  value: 検索する文字列。
  返値: 指定した文字列が見つかった場合は0から始まるインデックス、見つからなかった場合は-1。

- bool startsWith(string s, string value)
  元の文字列が、指定した文字列で始まっているかどうかを検査します。
  引数:
  s: 元となる文字列。
  value: 比較する文字列。
  返値: 元の文字列が指定した文字列で始まっている場合はtrue、そうでない場合はfalse。

◆ bool endsWith(string s, string value)
元の文字列が、指定した文字列で終わっているかどうかを検査します。
引数:
s: 元となる文字列。
value: 比較する文字列。
返値: 元の文字列が指定した文字列で終わっている場合はtrue、そうでない場合はfalse。

◆ string replace(string s, string oldValue, string newValue)
元となる文字列から、指定した文字列を別の文字列に置換した新しい文字列を返します。
引数:
s: 元となる文字列。
oldValue: 置換対象の文字列。
newValue: 置換する文字列。
返値: 置換対象の文字列をすべて置換した新しい文字列。

◆ string trim(string s)
文字列の両端の空白文字をすべて削除します。
引数:
s: 両端の空白文字を削除する文字列。
返値: 両端から空白文字を削除された文字列。

◆ string substring(string s, double startIndex)
string substring(string s, double startIndex, double length)
文字列を指定した位置から末尾、または指定して長さまで切り出します。
引数:
s: 元となる文字列。
startIndex: 切り出しの開始位置。この値の小数点以下は切り捨てられます。
length: 切り出す文字列の長さ。この値の小数点以下は切り捨てられます。
返値: 切り出した文字列。

- string toUpper(string s)
  文字列をすべて大文字に変換します。
  引数:
  s: 変換する文字列。
  返値: 変換された文字列。

- string toLower(string s)
  文字列をすべて小文字に変換します。
  引数:
  s: 変換する文字列。
  返値: 変換された文字列。

- string toString(object value)
  指定した値を文字列に変換します。
  引数:
  value: 変換する値。
  返値: 変換された文字列。

- double toDouble(object value)
  指定した値を実数に変換します。
  引数:
  value: 変換する値。
  返値: 変換された実数。

- bool toBool(object value)
  指定した値を真偽値に変換します。
  引数:
  value: 変換する値。
  返値: 変換された真偽値。

- Vertex toVertex(object value)
  指定した値をベクトルに変換します。
  引数:
  value: 変換する値。
  返値: 変換されたベクトル。

◆ bool isDouble(object value)

指定した値が実数であるかどうかを検査します。

引数:

value: 検査する値。

返値: 値が実数である場合はtrue、そうでない場合はfalse。

◆ bool isString(object value)

指定した値が文字列であるかどうかを検査します。

引数:

value: 検査する値。

返値: 値が文字列である場合はtrue、そうでない場合はfalse。

◆ bool isBool(object value)

指定した値が真偽値であるかどうかを検査します。

引数:

value: 検査する値。

返値: 値が真偽値である場合はtrue、そうでない場合はfalse。

◆ bool isVertex(object value)

指定した値がベクトルであるかどうかを検査します。

引数:

value: 検査する値。

返値: 値がベクトルである場合はtrue、そうでない場合はfalse。

◆ double ease(double time1, double value1, double time2, double value2, double time, double ease)
Vertex ease(double time1, Vertex value1, double time2, Vertex value2, double time, double ease)
ExpressionUtils.Easeを呼び出します。
引数:
frame1: 開始するフレーム位置。
value1: 開始時の値。
frame2: 終了するフレーム位置。
value2: 終了時の値。
time: 現在のフレーム位置。
ease: 実数を入力します。1 で等速、1 未満の時は減速、1 以上の時は加速します。
返値: 計算された値。
返値:計算された値。

◆ double linear(double value1, double value2, double x)
Vertex linear(Vertex value1, double value2, double x)
Vertex linear(double value1, Vertex value2, double x)
Vertex linear(double value1, double value2, Vertex x)
Vertex linear(Vertex value1, Vertex value2, double x)
Vertex linear(Vertex value1, double value2, Vertex x)
Vertex linear(double value1, Vertex value2, Vertex x)
Vertex linear(Vertex value1, Vertex value2, Vertex x)
2つの値を線形補完します。
引数:
value1: 値の最小値。
value2: 値の最大値。
x: 0～1の間の重み。
返値: 補完された値。

◆ double smoothStep(double value1, double value2, double x)
Vertex smoothStep(Vertex value1, double value2, double x)
Vertex smoothStep(double value1, Vertex value2, double x)
Vertex smoothStep(double value1, double value2, Vertex x)
Vertex smoothStep(Vertex value1, Vertex value2, double x)
Vertex smoothStep(Vertex value1, double value2, Vertex x)
Vertex smoothStep(double value1, Vertex value2, Vertex x)
Vertex smoothStep(Vertex value1, Vertex value2, Vertex x)
ExpressionUtils.SmoothStepを呼び出します。
引数:
value1: 値の最小値。
value2: 値の最大値。
x: 0～1の間の重み。
返値: 補完された値。

◆ double wiggle(double x, double oct, double value, double amp)
Vertex wiggle(double x, double oct, Vertex value, double amp)
Vertex wiggle(double x, double oct, double value, Vertex amp)
Vertex wiggle(double x, double oct, Vertex value, Vertex amp)
ExpressionUtils.Wiggleを呼び出します。seedの値はseedRandomによって設定されたシードを使用します。
引数:
x: 0～360 までの展開。
oct: 0～100 までの振動の複雑性。小数点以下は切り捨てられます。
value: 元となる値。
amp: 振動の振幅。
返値: 取得した値。

◆ object getProperty(string propertyName)
object getProperty(string layerName, string effectName, string propertyName)
現在のレイヤー、またはエフェクト、もしくは指定したアイテムからプロパティを取得します。
引数:
layerName: 対象のレイヤー名。現在のレイヤーを指定する場合はnull、もしくは空文字。
effectName: 対象のエフェクト名。現在のエフェクト、またはレイヤープロパティを指定する場合はnull、もしくは空文字。
propertyName: 取得するプロパティの名前。
返値: 取得したプロパティ。レイヤーやエフェクト、プロパティが存在しない場合はnull。

◆ double timeToFrames(double time)
double timeToFrames(double time, bool isDuration)
double timeToFrames(double time, double frameRate)
double timeToFrames(double time, double frameRate, bool isDuration)
タイムベースの時間を現在のコンポジションのフレームレート、または指定したフレームレートでのフレーム数に変換します。
引数:
time: タイムベースの時間。
frameRate: 変換時のフレームレート。
isDuration: これは、フレーム数計算時に発生した小数点以下の丸め方の違いを示します。timeが絶対時間である場合はfalse、相対時間である場合はtrueを指定します。trueの場合はceil、falseの場合はfloorが適用されます。
返値: 変換されたフレーム数。

◆ double framesToTime(double frames)
double framesToTime(double frames, double frameRate)
フレーム数から、現在のコンポジションのフレームレート、または指定したフレームレートでのタイムベースの時間に変換します。
引数:
frames: フレーム数。
frameRate: 変換時のフレームレート。
返値: 変換されたタイムベースの時間。

◆ string timeToTimecode(double time)
string timeToTimecode(double time, bool isDuration)
string timeToTimecode(double time, double frameRate)
string timeToTimecode(double time, double frameRate, bool isDuration)
タイムベースの時間を現在のコンポジションのフレームレート、または指定したフレームレートでのタイムコード表記に変換します。引数：
time：タイムベースの時間。
frameRate：変換時のフレームレート。
isDuration：timeToFramesを参照してください。
返値：変換されたタイムコード。

◆ object getEffectInfo(string infoName)
object getEffectInfo(string effectName, string infoName)
object getEffectInfo(string layerName, string effectName, string infoName)
現在のエフェクト、または指定したエフェクトの情報を取得します。
引数：
layerName：対象のレイヤー名。現在のレイヤーを指定する場合はnull、もしくは空文字。
effectName：対象のエフェクト名。現在のエフェクト、またはレイヤープロパティを指定する場合はnull、もしくは空文字。
infoName：取得する情報の名前を指定します。取得可能な情報は以下の通りです。
effectName：エフェクトのプラグイン名。
itemName：エフェクトの名前。
enable：エフェクトの有効状態。
turn：エフェクトの順番。
返値：指定した情報。エフェクトやレイヤー、情報が存在しない場合はnull。

◆ object getLayerInfo(string infoName)
object getLayerInfo(string layerName, string infoName)
現在のレイヤー、または指定したレイヤーの情報を取得します。
引数:
layerName: 対象のレイヤー名。現在のレイヤーを指定する場合はnull、もしくは空文字。
infoName: 取得する情報の名前を指定します。取得可能な情報は以下の通りです。
sourceName: 追加時のレイヤーのソース名。
itemName: レイヤーの名前。
haveVideo: レイヤーがビデオを持つかどうか。
haveAudio: レイヤーがオーディオを持つかどうか。
enable: レイヤーの有効状態。
enableVideo: レイヤーのビデオの有効状態。
enableAudio: レイヤーのオーディオの有効状態。
enableEffect: レイヤーのエフェクトスイッチの有効状態。
enable3D: レイヤーの3Dスイッチの有効状態。
enableMB: レイヤーのモーションブラースイッチの有効状態。
enableHQ: レイヤーの高画質スイッチの有効状態。
enableReverse: 逆再生の有効状態。
duration: レイヤーの長さ。
inPoint: レイヤーの開始時間。
outPoint: レイヤーの終了時間。
position: レイヤーの位置。
turn: レイヤーの順番。
返値: 指定した情報。レイヤーや情報が存在しない場合はnull。

◆ object getCompInfo(string infoName)

現在のコンポジションの情報を取得します。

引数:

infoName: 取得する情報の名前を指定します。取得可能な情報は以下の通りです。

itemName: コンポジションの名前。

enableFB: フレームブレンドスイッチの有効状態。

enableMB: モーションブラースイッチの有効状態。

length: コンポジションの長さ。

frameRate: コンポジションのフレームレート。

samplingMB: モーションブラーのサンプルレート。

shutterAngle: シャッター角度。

shutterPhase: シャッターフェーズ。

width: コンポジションの幅。

height: コンポジションの高さ。

返値: 指定した情報。情報が存在しない場合はnull。

◆ void seedRandom(double seed)

void seedRandom(double seed, double time)

乱数のシードを設定します。

引数:

seed: 乱数のシード値。

time: 時間により変化させる場合、ここに時間を代入します。

◆ double random()

double random(double maxValue)

double random(double minValue, double maxValue)

0～1、または最小値以上最大値以下の乱数を返します。

引数:

maxValue: 乱数の最大値。

minValue: 乱数の最小値。

返値: 0～1、または最小値以上最大値以下の乱数。

●エクスプレッション

■ 出来ること

エクスプレッションでは、同一コンポジション内のプロパティをレイヤー、エフェクトから操作することが出来ます。

■ エクスプレッションの仕組み

エクスプレッションは、主に次のプロパティを取得する時にすべてのエクスプレッションが評価されます。評価順序はレイヤー、エフェクトの順で行われます。

・レイヤー・エフェクトでのエクスプレッション

レンダリング実行時、ILayer.GetLayerProperty、ILayer.GetEffectProperty 実行時

・入力プロパティのエクスプレッション

入力プラグインにイメージ・ビデオを要求時、ILayer.GetRendererItemProperty 実行時

プロパティは、ExpressionPropertyContainer クラスから取得、または設定することが出来ます。エクスプレッション評価時には property 変数に与えられます。

■ 制限

レイヤー、エフェクトのエクスプレッションと、入力プロパティのエクスプレッションは、それぞれ全く別の別のタイミングで評価されるため、入力プロパティからレイヤー、エフェクトプロパティ、またはその逆を操作することは出来ません。

また、エクスプレッションでは、以下のキーワードがすでに使用されているため、エクスプレッション内では使用できません。

・timer
・Monitor
・Initialize
・Expression
・time
・property
・thisItem
・composition
・GetMonitor
・timer_Elapsed

NicoVusualEffects$_2$
コードの記述について

■ 使用方法

使用方法は、それぞれのコンテキストメニューから「エクスプレッションを使用する」を選択し、現れたエクスプレッションコントロールにコードを書き込みます。左側の「表示：」のところをクリックすることで、入力フィールドを切り替えることが出来ます。

・メイン

メインとなるコードを記述します。

・メソッド・フィールド

メソッドやフィールド、インナークラスはここに記述します。

・名前空間

デフォルトで宣言されている名前空間以外を使用する場合はここに記述します。改行区切りで、using キーワードは不要です。

・参照

デフォルトで参照するアセンブリ以外を参照する場合は、ここに入力してください。改行区切りで.NET クラスライブラリ以外のアセンブリはフルパスで入力してください。

・プロパティ

プロパティモニタに追加されたプロパティの値を表示します。

■ テキストファイルでのエクスプレッションコードのフォーマット

テキストファイルでは、以下のキーワードをブロックの開始行で指定することによって、コードを記述出来ます。それぞれのブロックでの記述ルールは、上記の使用方法に準じます。

・[MainCode]

メインに表示されるコードを示します。

・[MethodCode]

メソッド・フィールドに表示されるコードを示します。

・[UsingNamespace]

名前空間に表示されるコードを示します。

・[Reference]

参照に表示されるコードを示します。

■ 注意点

レイヤーやエフェクトのアイテムコードは、ペーストされる度に変更されます。特定のアイテムを参照する場合、出来る限りアイテム名で判別してください。

NicoVusualEffects$_2$
コードの記述について

■ ライブラリ

IExpressionItem や、ExpressionUtils について記述します。

■ interface IExpression

エクスプレッションのコードを実行するクラスに継承されます。このインターフェースはユーザーが使用することはありません。

◆ void Expression(double time, ExpressionPropertyContainer property, IExpressionItem thisItem, IExpressionComposition[] composition)

エクスプレッションを実行します。

引数:

time: 実行時の時間

property: レイヤー、エフェクトなどのプロパティ。

thisItem: 実行対象の IExpressionItem。

composition: すべてのコンポジション。

■ interface IExpressionItem

エクスプレッション内で使用するアイテムのインターフェースです。

◆ int ItemCode

アイテムコードを取得します。

◆ string ItemName

アイテムの名前を取得します。

◆ IExpressionItem ParentItem

このアイテムの親となるアイテムを取得します。

◆ int Turn

このアイテムの順番を取得します。

■ interface IExpressionLayer : IExpressionItem

エクスプレッション内で使用するレイヤーのインターフェースです。

◆ IExpressionLayer ParentExpressionLayer

このレイヤーの親レイヤーを取得します。

NicoVusualEffects$_2$
コードの記述について

■ `interface IExpressionComposition : IExpressionItem`
エクスプレッション内で使用するコンポジションのインターフェースです。

◆ `ExpressionPropertyContainer ExecuteExpression(double time)`
このコンポジションのエクスプレッション評価済みのプロパティを取得します。

■ `static class ExpressionUtils`
エクスプレッションを実行する上で有効だと思われる処理をまとめたクラスです。

◆ `static void WriteText(string file, string text, bool append)`
テキストを書き出します。
引数:
file: 書き出し先のファイル。
text: 書き込むテキスト。
append:追記する場合は true、上書きする場合は false。

◆ `static string ReadText(string file)`
テキストを読み込みます。
引数:
file: 読み込むテキストファイル。
返値: 読み込んだテキスト。

◆ static int Ease(double frame1, int value1, double frame2, int value2, double time, double ease)
static float Ease(double frame1, float value1, double frame2, float value2, double time, double ease)
static double Ease(double frame1, double value1, double frame2, double value2, double time, double ease)
static Point Ease(double frame1, Point value1, double frame2, Point value2, double time, double ease)
static PointF Ease(double frame1, PointF value1, double frame2, PointF value2, double time, double ease)
static Size Ease(double frame1, Size value1, double frame2, Size value2, double time, double ease)
static SizeF Ease(double frame1, SizeF value1, double frame2, SizeF value2, double time, double ease)
static Color Ease(double frame1, Color value1, double frame2, Color value2, double time, double ease)
static Vertex Ease(double frame1, Vertex value1, double frame2, Vertex value2, double time, double ease)
開始値から終了値に向けて徐々に加速したり減速する関数です。
引数:
frame1: 開始するフレーム位置。
value1: 開始時の値。
frame2: 終了するフレーム位置。
value2: 終了時の値。
time: 現在のフレーム位置。
ease: 実数を入力します。1 で等速、1 未満の時は減速、1 以上の時は加速します。
返値: 計算された値。

◆ static int SmoothStep(int edge1, int edge2, double value)
static float SmoothStep(float edge1, float edge2, double value)
static double SmoothStep(double edge1, double edge2, double value)
static Point SmoothStep(Point edge1, Point edge2, double value)
static PointF SmoothStep(PointF edge1, PointF edge2, double value)
static Size SmoothStep(Size edge1, Size edge2, double value)
static SizeF SmoothStep(SizeF edge1, SizeF edge2, double value)
static Color SmoothStep(Color edge1, Color edge2, double value)
static Vertex SmoothStep(Vertex edge1, Vertex edge2, double value)
2つの値をエルミート補完します。
引数:
value1: 値の最小値。
value2: 値の最大値。
x: 0～1の間の重み。
返値: 補完された値。

◆ static int Wiggle(int seed, int oct, int value, int amp)
static int Wiggle(int seed, float x, int oct, int value, int amp)
static float Wiggle(int seed, int oct, float value, float amp)
static float Wiggle(int seed, float x, int oct, float value, float amp)
static double Wiggle(int seed, int oct, double value, double amp)
static double Wiggle(int seed, float x, int oct, double value, double amp)
static Point Wiggle(int seed, int oct, Point value, Point amp)
static Point Wiggle(int seed, float x, int oct, Point value, Point amp)
static PointF Wiggle(int seed, int oct, PointF value, PointF amp)
static PointF Wiggle(int seed, float x, int oct, PointF value, PointF amp)
static Size Wiggle(int seed, int oct, Size value, Size amp)
static Size Wiggle(int seed, float x, int oct, Size value, Size amp)
static SizeF Wiggle(int seed, int oct, SizeF value, SizeF amp)
static SizeF Wiggle(int seed, float x, int oct, SizeF value, SizeF amp)
static Vertex Wiggle(int seed, int oct, Vertex value, Vertex amp)
static Vertex Wiggle(int seed, float x, int oct, Vertex value, Vertex amp)
値を一定の範囲内で振動させます。
引数:
seed: 乱数のシード値。
x: 0～360 までの展開。
oct: 0～100 までの振動の複雑性。
value: 元となる値。
amp: 振動の振幅。
返値: 取得した値。

■ abstract class ExpressionPropertyContainer

レイヤーやエフェクト、入力プロパティを格納するクラスです。

◆ string[] GetPropertyNames(IExpressionItem identify)
string[] GetPropertyNames(ILayer identify)
string[] GetPropertyNames(IEffect identify)
レイヤーやエフェクト、入力プロパティの名前を取得します。
引数:
identify: 取得するアイテム。
返値: 取得したプロパティの名前の配列。

◆ PropertyBase[] GetProperties(IExpressionItem identify)
PropertyBase[] GetProperties(ILayer identify)
PropertyBase[] GetProperties(IEffect identify)
指定したアイテムのすべてのプロパティを取得します。
引数:
identify: 取得するアイテム。
返値: 取得したプロパティの配列。

◆ PropertyBase GetProperty(IExpressionItem identify, string propertyName)
PropertyBase GetProperty(ILayer identify, string propertyName)
PropertyBase GetProperty(IEffect identify, string propertyName)
指定したアイテムのプロパティを取得します。
引数:
identify: 取得するアイテム。
propertyName: 取得するプロパティの名前。
返値: 取得したプロパティ。

◆ void SetProperty(IExpressionItem identify, PropertyBase property)
void SetProperty(ILayer identify, PropertyBase property)
void SetProperty(IEffect identify, PropertyBase property)
指定したアイテムのプロパティを設定します。
引数:
identify: 設定するアイテム。
property: 設定するプロパティ。

- `ExpressionPropertyContainer Copy()`
  このインスタンスの完全コピーを返します。
  返値：コピーされた ExpressionPropertyContainer。

●オートメーション

■ 出来ること

現在、オートメーションでは、
・各種レイヤースイッチの変更
・レイヤー・エフェクトのリネーム
・レイヤー・エフェクトのコピー・切り取り・貼り付け
・レイヤーの分割
・レイヤーのデュレーション・逆再生の変更
・コンポジションの設定の変更
・ファイルの読み込み
・カラーイメージ・コンポジションの作成
・コンポジションへのアイテムの追加
・レイヤーへのエフェクトの追加
・プロパティ・キーフレームの設定
が可能です。
プロパティの設定は、可変長プロパティには対応していません。

■ オートメーションの仕組み

オートメーションは、基本的に `IAutomationManager` の機能によって実現されます。
`IAutomationManager` を通して行った操作は、自動的にヒストリーにも追加され、
Undo/Redo が可能です。

■ 制限

オートメーションでは、次のキーワードがすでに使用されているため、オートメーションコード中では使用できません。
・timer
・Execute
・timer_Elapsed

NicoVusualEffects$_2$
コードの記述について

■ 使用方法

使用方法は、メインメニューの「編集」から「オートメーション」(Ctrl + M)を選択し、コードを記述します。各タブの役割は以下の通りです。

・メインタブ

メインとなるコードを記述します。IAutomationManager へは manager 変数よりアクセスできます。

・メソッド・フィールドタブ

メソッドやフィールド、インナークラスはここに記述します。

・名前空間タブ

デフォルトで宣言されている名前空間以外を使用する場合はここに記述します。改行区切りで、using キーワードは不要です。

・参照タブ

デフォルトで参照するアセンブリ以外を参照する場合は、ここに入力してください。改行区切りで.NET クラスライブラリ以外のアセンブリはフルパスで入力してください。

・保存ボタン

オートメーションコードを、NiVE Automation Code ファイル(*.nvac)、またはテキストファイル(*.txt)として保存します。

・読み込みボタン

nvac ファイル、またはテキストファイルからから、オートメーションコードを読み込みます。

■ テキストファイルでのオートメーションコードのフォーマット

テキストファイルでは、以下のキーワードをブロックの開始行で指定することによって、コードを記述出来ます。それぞれのブロックでの記述ルールは、上記の使用方法に準じます。

・[MainCode]

メインタブに表示されるコードを示します。

・[MethodCode]

メソッド・フィールドタブに表示されるコードを示します。

・[UsingNamespace]

名前空間タブに表示されるコードを示します。

・[Reference]

参照タブに表示されるコードを示します。

NicoVusualEffects$_2$
コードの記述について

■ ライブラリ

IAutomationManager や、その周辺クラスについて記述します。

■ enum ResolutionRateMode

プレビューの画質を表します。

◆ Full

フル画質。

◆ Half

1/2 画質。

◆ Third

1/3 画質。

◆ Quarter

1/4 画質。

◆ HalfQuarter

1/8 画質。

■ interface IAutomationManager

オートメーションを実行するための機能を提供します。

◆ void ChangeEnableVideo(ILayer layer, bool enable)
void ChangeEnableVideo(ILayer[] layer, bool enable)

ビデオ、またはレイヤーの有効状態を変更します。

引数:

layer: 変更するレイヤー。

enable: 変更後の状態。

- `void ChangeEnableAudio(ILayer layer, bool enable)`
  `void ChangeEnableAudio(ILayer[] layer, bool enable)`
  オーディオの有効状態を変更します。
  引数:
  layer: 変更するレイヤー。
  enable: 変更後の状態。

- `void ChangeEnableShy(ILayer layer, bool enable)`
  `void ChangeEnableShy(ILayer[] layer, bool enable)`
  レイヤーのシャイの有効状態を変更します。
  引数:
  layer: 変更するレイヤー。
  enable: 変更後の状態。

- `void ChangeEnableHighRenderingQuality(ILayer layer, bool enable)`
  `void ChangeEnableHighRenderingQuality(ILayer[]layer, bool enable)`
  レイヤーの高画質レンダリングの有効状態を変更します。
  引数:
  layer: 変更するレイヤー。
  enable: 変更後の状態。

- `void ChangeEnableEffect(ILayer layer, bool enable)`
  `void ChangeEnableEffect(ILayer[] layer, bool enable)`
  レイヤーのエフェクトを適用するかどうかを変更します。
  引数:
  layer: 変更するレイヤー。
  enable: 変更後の状態。

- `void ChangeEnableMotionBlur(ILayer layer, bool enable)`
  `void ChangeEnableMotionBlur(ILayer[] layer, bool enable)`
  レイヤーにモーションブラーを適用するかどうかを変更します。
  引数:
  layer: 変更するレイヤー。
  enable: 変更後の状態。

◆ void ChangeEnable3D(ILayer layer, bool enable)
void ChangeEnable3D(ILayer[] layer, bool enable)
レイヤーを 3D レイヤーにするかどうかを変更します。
引数:
layer: 変更するレイヤー。
enable: 変更後の状態。

◆ void ChangeFrameBlendMode(ILayer layer, FrameBlendMode blendMode)
void ChangeFrameBlendMode(ILayer[] layer, FrameBlendMode blendMode)
レイヤーのフレームブレンドモードを変更します。
引数:
layer: 変更するレイヤー。
blendMode: 変更後のフレームブレンドモード。

◆ void ChangeEnableEffect(IEffect effect, bool enable)
void ChangeEnableEffect(IEffect[] effect, bool enable)
エフェクトの有効状態を変更します。
引数:
effect: 変更するエフェクト。
enable: 変更後の状態。

◆ void ChangeLayerName(ILayer layer, string name)
レイヤーの名前を変更します。
引数:
layer: 変更するレイヤー。
name: 変更後のレイヤーの名前。

◆ void ChangeEffectName(IEffect effect, string name)
エフェクトの名前を変更します。
引数:
effect: 変更するエフェクト。
name: 変更後のエフェクトの名前。

- `void ChangeDuration(ILayer layer, double time, bool reverse)`
  レイヤーの長さを変更します。
  引数:
  layer: 変更するレイヤー。
  time: 変更後の長さ。
  reverse: 逆再生する場合は true、そうでない場合は false を指定します。

- `void ChangeLayerTime(ILayer layer, double time)`
  `void ChangeLayerTime(ILayer[] layer, double time)`
  レイヤー時間を変更します。
  引数:
  layer: 変更するレイヤー。
  time: 変更後の時間。

- `void ChangeInPoint(ILayer layer, double time)`
  `void ChangeInPoint(ILayer[] layer, double time)`
  レイヤーの開始時間を変更します。
  引数:
  layer: 変更するレイヤー。
  time: 変更後の時間。

- `void ChangeOutPoint(ILayer layer, double time)`
  `void ChangeOutPoint(ILayer[] layer, double time)`
  レイヤーの終了時間を変更します。
  引数:
  layer: 変更するレイヤー。
  time: 変更後の時間。

- `void ChangeCompositionSetting(IComposition composition, CompositionSetting setting)`
  コンポジションの設定を変更します。
  引数:
  composition: 設定を変更するコンポジション。
  setting: 変更後の設定。

- void ChangeShyCompositionSwitch(IComposition composition, bool enable)
  void ChangeShyCompositionSwitch(IComposition[] composition, bool enable)
  コンポジションのシャイスイッチを変更します。
  引数:
  composition: 変更するコンポジション。
  enable: 有効にする場合はtrue、そうでない場合はfalse。

- void ChangeFrameBlendCompositionSwitch(IComposition composition, bool enable)
  void ChangeFrameBlendCompositionSwitch(IComposition[] composition, bool enable)
  コンポジションのフレームブレンドスイッチを変更します。
  引数:
  composition: 変更するコンポジション。
  enable: 有効にする場合はtrue、そうでない場合はfalse。

- void ChangeMotionBlurCompositionSwitch(IComposition composition, bool enable)
  void ChangeMotionBlurCompositionSwitch(IComposition[] composition, bool enable)
  コンポジションのモーションブラースイッチを変更します。
  引数:
  composition: 変更するコンポジション。
  enable: 有効にする場合はtrue、そうでない場合はfalse。

- void ChangeResolutionRate(IComposition composition, ResolutionRateMode mode)
  void ChangeResolutionRate(IComposition[] composition, ResolutionRateMode mode)
  コンポジションの画質モードを変更します。
  引数:
  composition: 変更するコンポジション。
  mode: 設定する画質。

- `void ChangeParent(ILayer layer, ILayer newParent)`
  `void ChangeParent(ILayer layer, ILayer newParent, double setTime)`
  `void ChangeParent(ILayer[] layer, ILayer newParent)`
  `void ChangeParent(ILayer[] layer, ILayer newParent, double setTime)`
  レイヤーの親を変更します。
  引数:
  `layer`: 変更するレイヤー、またはその配列。
  `newParent`: 新しい親となるレイヤー。親を解除する場合は `null`。
  `setTime`: 親を設定する時間。親が `null` の場合は無視する。

- `IComposition[] GetComposition()`
  全てのコンポジションを取得します。
  返値: 取得したコンポジション。

- `InputChild[] GetInputChild()`
  全てのアイテムを取得します。
  返値: 取得したアイテム。

- `ReadOnlyDictionary<string, Type> GetAllRendererPlugin()`
  プラグインの完全修飾名をキーとするレンダラプラグインのディクショナリを取得します。
  返値: レンダラプラグインの読み取り専用ディクショナリ。

- `Type GetRendererPlugin(string pluginName)`
  プラグイン名からレンダラプラグインを検索します。
  引数:
  `name`: 検索するプラグイン名。
  返値: 存在する場合はプラグインの型、そうでない場合は `null` を返します。

- `ReadOnlyDictionary<string, Type> GetAllInputPlugin()`
  プラグインの完全修飾名をキーとする入力プラグインのディクショナリを取得します。
  返値: 入力プラグインの読み取り専用ディクショナリ。

- `Type GetInputPlugin(string pluginName)`
  プラグイン名から入力プラグインを検索します。
  引数:
  name: 検索するプラグイン名。
  返値: 存在する場合はプラグインの型、そうでない場合は null を返します。

- `ReadOnlyDictionary<string, Type> GetAllEffectPlugin()`
  プラグインの完全修飾名をキーとするエフェクトプラグインのディクショナリを取得します。
  返値: エフェクトプラグインの読み取り専用ディクショナリ。

- `Type GetEffectPlugin(string pluginName)`
  プラグイン名からエフェクトプラグインを検索します。
  引数:
  name: 検索するプラグイン名。
  返値: 存在する場合はプラグインの型、そうでない場合は null を返します。

- `InputChild[] LoadFile(string itemDirectory, Type plugin, string[] file)`
  ファイルを読み込みます。
  引数:
  itemDirectory: アイテムの追加先のディレクトリ。
  plugin: 読み込みに使用する入力プラグイン。
  file: 読み込むファイル。
  返値: 読み込まれたアイテム。

- `InputChild[] NewColorImage(string itemDirectory, Color color, Size size)`
  カラーイメージを読作成します。
  引数:
  itemDirectory: アイテムの追加先のディレクトリ。
  color: 読み込むカラーイメージの色。
  size: 読み込むカラーイメージのサイズ。
  返値: 読み込まれたアイテム。

NicoVusualEffects$_2$
コードの記述について

- `InputChild[] NewComposition(string itemDirectory, CompositionSetting setting)`
  コンポジションを作成します。
  引数:
  itemDirectory: アイテムの追加先のディレクトリ。
  setting: 作成するコンポジションの設定。
  返値: 読み込まれたアイテム。

- `ILayer AddLayer(IComposition composition, InputChild input, string sourceName)`
  コンポジションにレイヤーを追加します。
  引数:
  composition: 追加先のコンポジション。
  input: 追加するアイテム。
  sourceName: アイテムの基本名。
  返値: 追加されたレイヤー。

- `ILayer AddText(IComposition composition)`
  コンポジションにテキストレイヤーを追加します。
  引数:
  composition: 追加先のコンポジション。
  返値: 追加されたレイヤー。

- `ILayer AddShape(IComposition composition)`
  コンポジションにシェイプレイヤーを追加します。
  引数:
  composition: 追加先のコンポジション。
  返値: 追加されたレイヤー。

- `ILayer AddCamera(IComposition composition)`
  コンポジションにカメラレイヤーを追加します。
  引数:
  composition: 追加先のコンポジション。
  返値: 追加されたレイヤー。

NicoVusualEffects$_2$
コードの記述について

- `ILayer AddNullObject(IComposition composition)`
  コンポジションに Null オブジェクトレイヤーを追加します。
  引数:
  composition: 追加先のコンポジション。
  返値: 追加されたレイヤー。

- `ILayer AddLight(IComposition composition)`
  コンポジションにライトレイヤーを追加します。
  引数:
  composition: 追加先のコンポジション。
  返値: 追加されたレイヤー。

- `IEffect AddEffect(ILayer layer, Type effect)`
  レイヤーにエフェクトを追加します。
  引数:
  layer: 追加先のレイヤー。
  effect: 追加するエフェクトプラグイン。
  返値: 追加されたエフェクト。

- `LayerData CopyLayer(ILayer layer)`
  `LayerData[] CopyLayer(ILayer[] layer)`
  レイヤーをコピーします。
  引数:
  layer: コピーするレイヤー。
  返値: コピーしたレイヤーのデータ。

- `LayerData CutLayer(ILayer layer)`
  `LayerData[] CutLayer(ILayer[] layer)`
  レイヤーを切り取ります。
  引数:
  layer: 切り取るレイヤー。
  返値:切り取ったレイヤーのデータ。

◆ `ILayer PasteLayer(IComposition composition, LayerData data)`
`ILayer[] PasteLayer(IComposition composition, LayerData[] data)`
コンポジションにレイヤーを貼り付けます。
引数:
composition: 貼り付け先のコンポジション。
data: 貼り付けるレイヤーのデータ。
返値: 貼り付けられたレイヤー。

◆ `ILayer DivideLayer(ILayer layer, double divideTime)`
`ILayer[] DivideLayer(ILayer[] layer, double divideTime)`
レイヤーを分割します。
引数:
layer: 分割するレイヤー。
divideTime: 分割するレイヤーのローカル時間。
返値: 分割されたレイヤー。

◆ `EffectData CopyEffect(IEffect effect)`
`EffectData[] CopyEffect(IEffect[] effect)`
エフェクトをコピーします。
引数:
effect: コピーするエフェクト。
返値: コピーしたエフェクトのデータ。

◆ `EffectData CutEffect(IEffect effect)`
`EffectData[] CutEffect(IEffect[] effect)`
エフェクトを切り取ります。
引数:
effect: 切り取るエフェクト。
返値: 切り取ったエフェクトのデータ。

◆ IEffect PasteEffect(ILayer layer, EffectData data)
IEffect[] PasteEffect(ILayer layer, EffectData[] data)
エフェクトを貼り付けます。
引数:
layer: 貼り付け先のレイヤー。
data: 貼り付けるエフェクトのデータ。
返値: 貼り付けたエフェクト。

◆ string[] GetPropertyNames(ILayer layer)
string[] GetPropertyNames(IEffect effect)
レイヤー、またはエフェクトのプロパティの名前を取得します。
引数:
layer: 取得対象のレイヤー。
effect: 取得対象のエフェクト。
返値: 取得したプロパティの名前の string 配列。

◆ bool GetMakeKeyFrame(ILayer layer, string propertyName)
bool GetMakeKeyFrame(IEffect effect, string propertyName)
キーフレームを作成するかどうかの値を取得します。
引数:
layer: 取得先のプロパティの存在するレイヤー。
effect: 取得先のプロパティの存在するエフェクト。
propertyName: 取得するプロパティ名。
返値: キーフレームを作成する場合は true、そうでない場合は false。

◆ void ChangeMakeKeyFrame(ILayer layer, string propertyName, bool enable)
void ChangeMakeKeyFrame(IEffect effect, string propertyName, bool enable)
キーフレームを作成するかどうかの値を変更します。
引数:
layer: 取得先のプロパティの存在するレイヤー。
effect: 取得先のプロパティの存在するエフェクト。
propertyName: 取得するプロパティ名。
enable: キーフレームを作成する場合は true、そうでない場合は false。

NicoVusualEffects$_2$
コードの記述について

◆ `PropertyBase GetProperty(ILayer layer, string propertyName, double time)`
`PropertyBase GetProperty(IEffect effect, string propertyName, double time)`
プロパティを取得します。
引数:
layer: 取得先のプロパティの存在するレイヤー。
effect: 取得先のプロパティの存在するエフェクト。
propertyName: 取得するプロパティ名。
time: 取得する時間。
返値: 取得したプロパティ。

◆ `void SetProperty(ILayer layer, PropertyBase property, double time)`
`void SetProperty(IEffect effect, PropertyBase property, double time)`
レイヤー、またはエフェクトのプロパティを設定します。MakeKeyFrame が true の場合はキーフレームを作成、または変更します。
引数:
layer: 設定先のレイヤー。
effect: 設定先のエフェクト。
property: 設定するプロパティ。
time: 設定する時間。

◆ `KeyFrame FindKeyFrame(ILayer layer, string propertyName, Predicate<KeyFrame> match)`
`KeyFrame FindKeyFrame(IEffect effect, string propertyName, Predicate<KeyFrame> match)`
キーフレームを検索します。
引数:
layer: 検索先のレイヤー。
effect: 検索先のエフェクト。
propertyName: 検索するプロパティ。
match: 検索条件。
返値: 見つかった場合はその KeyFrame、見つからなかった場合は null。

- ◆ `void AddKeyFrame(ILayer layer, KeyFrame keyFrame)`
  `void AddKeyFrame(IEffect effect, KeyFrame keyFrame)`
  レイヤー、またはエフェクトのプロパティにキーフレームを追加します。
  引数:
  layer: 設定先のレイヤー。
  effecct: 設定先のエフェクト。
  kayFrame: 設定するキーフレーム。

- ◆ `void RemoveKeyFrame(ILayer layer, string propertyName, double time)`
  `void RemoveKeyFrame(IEffect effect, string propertyName, double time)`
  指定した時間にあるキーフレームを削除します。
  引数:
  layer: 削除先のレイヤー。
  effect: 削除先のエフェクト。
  propertyName: 削除するキーフレームのプロパティ名。
  time: 削除するキーフレームの時間。

●更新履歴

- 2.00 α
  リリース
- 2.00 α3
  オートメーションのデフォルトの名前空間に NiVE2.Plugin.Property を追加
  GetPropertyNames、GetMakeKeyFrame、ChangeMakeKeyFrame、GetProperty、SetProperty、FindKeyFrame 、AddKeyFrame、RemoveKeyFrame を追加
- 2.00 α4
  ChangeParent を追加
- 2.00 α5
  エクスプレッションを追加
- 2.00 β2
  ChangeParent のオーバーロードを追加
- 2.00 β3
  AddShape メソッドを追加
- 2.00
  AddLight メソッドを追加
- 2.01
  ChangeLayerTime、ChangeInPoint、ChangeOutPoint メソッドを追加
- 2.02
  ChangeShyCompositionSwitch 、 ChangeFrameBlendCompositionSwitch 、ChangeMotionBlurCompositionSwitch、ChangeResolutionRate を追加
  テキストファイルでのエクスプレッションコードのフォーマットを追加
- 2.10
  プロパティスクリプトについてを追加
- 2.12
  プロパティスクリプトにベクトル型に関する部分を追加
  dot、cross、length、normalize、clamp、step、isVertex、toVertex、linear、smoothStep、各種ベクトル型対応のメソッドを追加
  ExpressionUtils.SmoothStep を追加