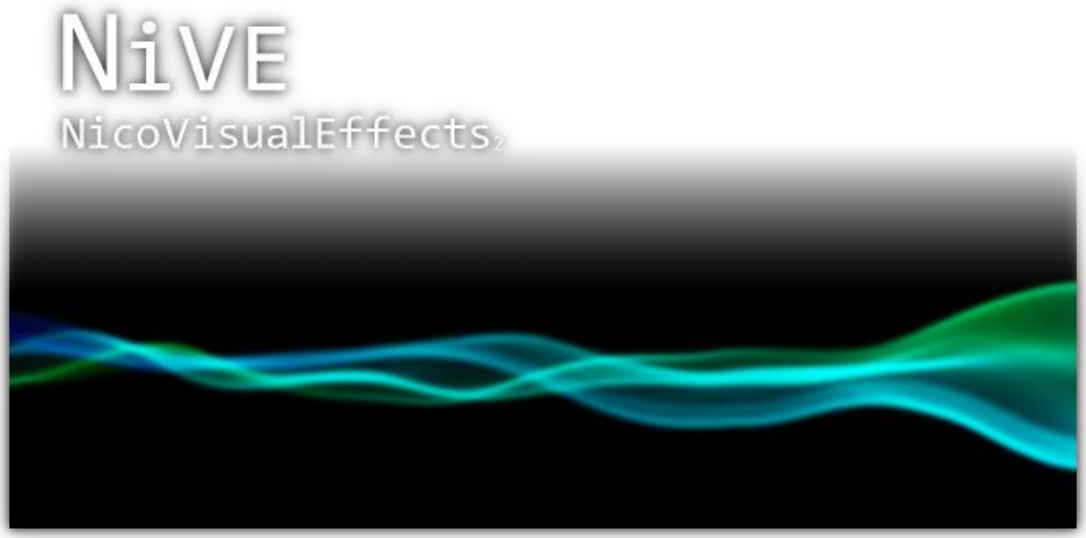

NicoVisualEffects2

エクスプレッション・オートメーションについて

●目次

●はじめに

エクスプレッションとオートメーションは、NiVE をプログラムから操作する機能です。

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。

これらの機能は言語に 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。

●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

■ 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。

■ 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。

■ 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。

NicoVusualEffects$_2$
エクスプレッション・オートメーションについて

■ 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。

■ 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 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

■ 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$_2$
エクスプレッション・オートメーションについて

■ ライブラリ

IAutomationManager で定義されているメソッドについて記述します。

◆ 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 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 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。

◆ 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。

◆ `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。

◆ `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 β 2
  ChangeParent のオーバーロードを追加
- 2.00 β 3
  AddShape メソッドを追加
- 2.00
  AddLight メソッドを追加