By NAGAOKA Shunsuke (GitHub@nagaoka-shusnuke;Twitter@sn_pluso)

2018-11-11 ver0.0

はじめに

Microsoft Visual Studio Community 2017 を用いて Grasshopper プラグインを開発する手順を解説する。プラグインを作成する流れを説明することを目的としているので、詳しい仕様やC#スクリプトの記述法について言及するものではないのでご了承いただきたい。また本資料の内容に基づく開発・運用結果について筆者は一切の責任を負いかねるのでご注意いただきたい。

開発するコンポーネントの内容

2 点を指定してその間を 1 周期の sin 波で補間するコンポーネントとする。
アルゴリズムなど詳細は後述する。

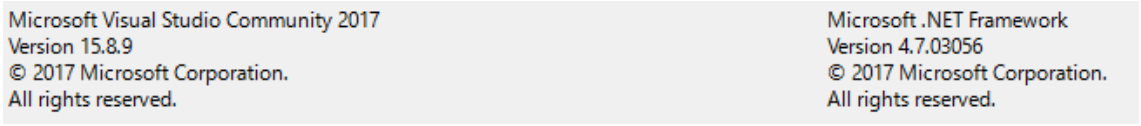
開発環境

○PC

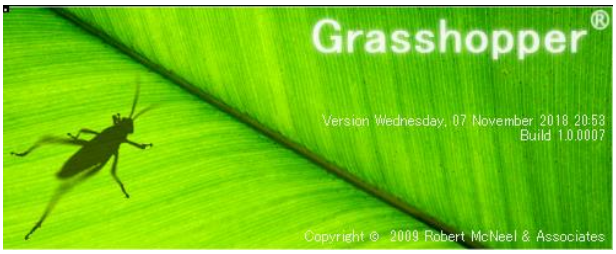
OS　：Windows 10 Pro version 1803
CPU：Intel®Core™ i7-8700CPU@3.20GHz 3.19GHz
RAM：16.0GB
GPU：NVIDIA GeForceGTX 1060 6GB

○Software

Microsoft Visual Studio Community 2017 Version 15.8.9
Microsoft.NET Framework Version 4.7.03056



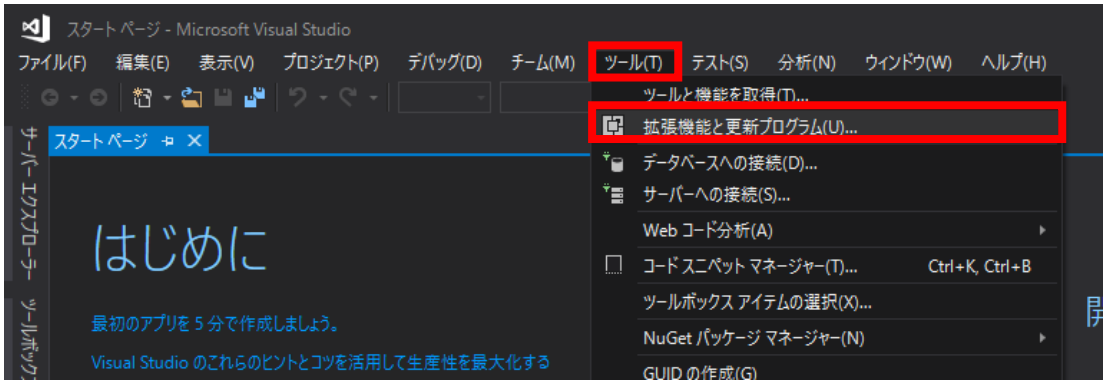
Rhinoceros Version 6 SR10(6.10.18311.20531, 11/07/2018)
Grasshopper Version Wednesday, 07 November 2018 20:53 Build 1.0.0007



Grasshopper コンポーネント開発アドオンのインストール

Visual Studio (VS)　を起動する。

プラグイン開発には Grasshopper Add-On for v6 という拡張機能を使用するが、これはデフォルトでインストールされていないので、オンラインから検索してダウンロードして利用する。初期画面が表示されたら、ツール＞拡張機能と更新プログラムからウィンドウを表示する。

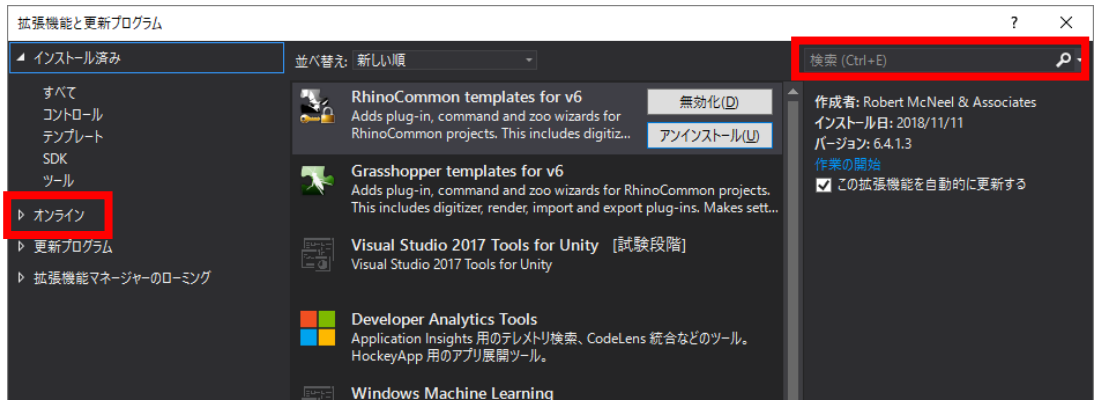


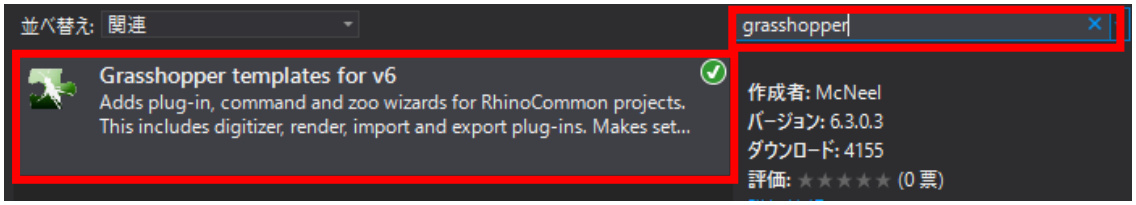
拡張機能と更新プログラムウィンドウが表示される。

左端のオンラインをクリックするとダウンロード可能な拡張機能の一覧が表示される。

右上に Grasshopper と打ち込み、Grasshopper templates for v6 をダウンロードする。

拡張機能の追加後は VS を再起動する必要がある旨のウィンドウが表示されるので、それに従って VS を再起動する。





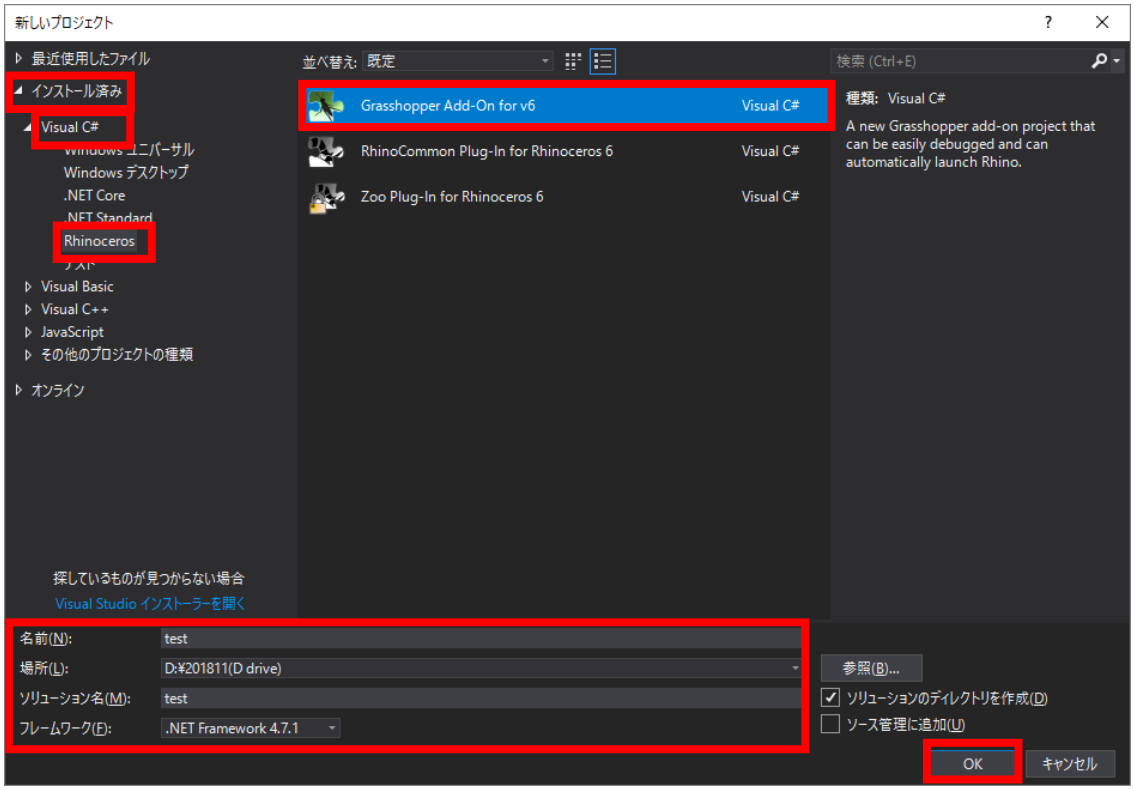
新規プロジェクトの作成

VS 初期画面から、ファイル＞新規作成＞プロジェクトで新しくプロジェクトを作成する。

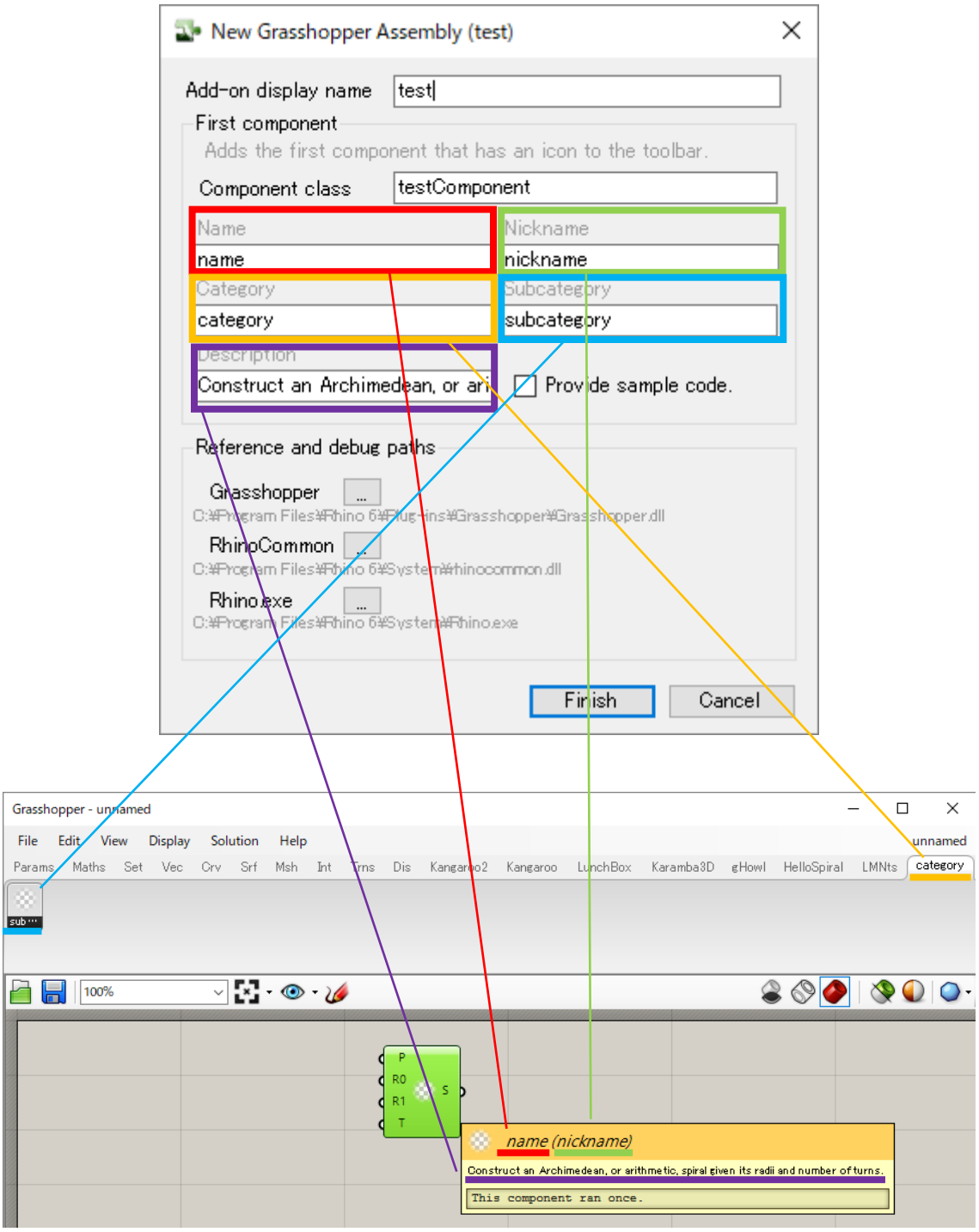
新規プロジェクトの作成ウィンドウが表示される。ここで先にインストールしたテンプレートの選択やプロジェクトに名前をつける操作を行う。左端のインストール済み＞Visual C#＞Rhinoceros とクリックし、表示された Grasshopper Add-On for v6 をクリックする。これで使用する拡張機能が選択できたことになる。

さらにプロジェクト・ソリューションの名前や保存場所、フレームワークを設定する。
ここでは名前を「test」、ソリューション名を「test」、フレームワークを.NET Framework 4.7.1 としている。
保存場所は適宜都合のいい場所に変更する。C ドライブである必要はないし、外部ストレージでもよい。その他は変更せず OK をクリックする。

ソリューション(.sln)は VS 独特の概念で、プロジェクトを格納する入れ物のようなものである。プロジェクト(.proj)はソリューションのなかに格納され、実行される。ここでは深く考えなくてもよい。
中断した作業を再開する際にはソリューション(.sln)ファイルを開くことになる。
フレームワークはプログラム開発時の共通機能のようなものである。

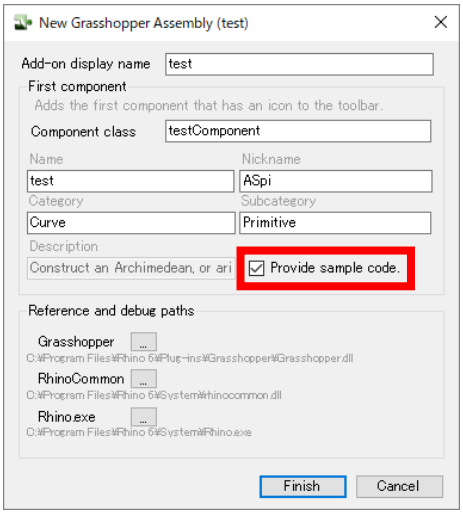


OK をクリックすると、New Grasshopper Assembly ウィンドウが表示され、コンポーネントの命名を行うことができる。このウィンドウで設定した名前がそのまま Grasshopper 上での表示名となる。入力と Grasshopper での表示の対応は次に示す通りである。

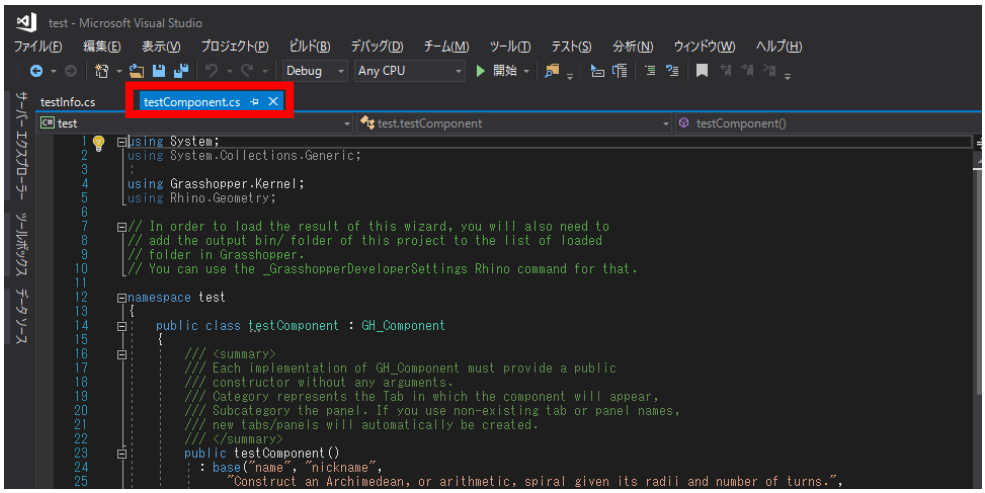


ここでは上記の通りに設定して Finish をクリックする。
コンポーネントのアイコン設定方法については後述する。

また、Provide sample code にチェックを入れてから Finish をクリックするとサンプルコードが読み込まれた状態で開始する。サンプルの中身は螺旋を描くコンポーネントである。今回はチェックをいれずに進めるが、コンポーネント作成上参考になるので確認してもらいたい。



これで新規プロジェクトの作成が終了した。
エディター画面が表示され、testInfo.cs と testComponent.cs が表示された状態となる。
これからはコンポーネント内部を記述する testComponent.cs にコードを書き加えていくことになる。



testComponent.cs の中身について

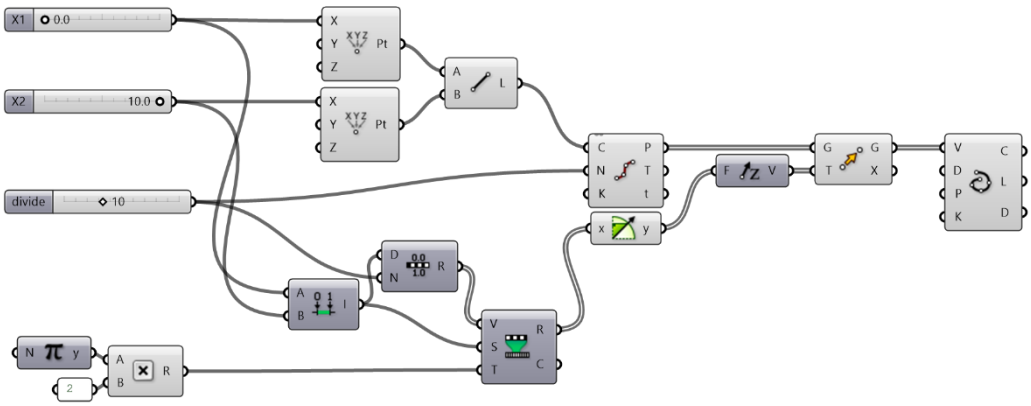
空の testComponent.cs 全文と簡単な説明を示す。
コンポーネントとして必要な要素がすべて記述されている。

```
using System;
using System.Collections.Generic;

using Grasshopper.Kernel;
using Rhino.Geometry;

// In order to load the result of this wizard, you will also need to
// add the output bin/ folder of this project to the list of loaded
// folder in Grasshopper.
// You can use the _GrasshopperDeveloperSettings Rhino command for that.

namespace test
{
    public class testComponent : GH_Component
    {
        /// <summary>
        /// Each implementation of GH_Component must provide a public
        /// constructor without any arguments.
        /// Category represents the Tab in which the component will appear,
        /// Subcategory the panel. If you use non-existing tab or panel names,
        /// new tabs/panels will automatically be created.
        /// </summary>
        public testComponent()
          : base("name", "nickname",
              "Construct an Archimedean, or arithmetic, spiral given its radii and number of turns.",
              "category", "subcategory")
        {                                                    コンポーネント名やカテゴリーを定義
        }

        /// <summary>
        /// Registers all the input parameters for this component.
        /// </summary>
        protected override void RegisterInputParams(GH_Component.GH_InputParamManager pManager)
        {                                                    コンポーネントのインプットを定義
        }

        /// <summary>
        /// Registers all the output parameters for this component.
        /// </summary>
        protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager pManager)
        {                                                    コンポーネントのアウトプットを定義
        }

        /// <summary>
        /// This is the method that actually does the work.
        /// </summary>
        /// <param name="DA">The DA object can be used to retrieve data from input parameters and
        /// to store data in output parameters.</param>
        protected override void SolveInstance(IGH_DataAccess DA)
        {                                                    コンポーネントの内部処理を定義
        }

        /// <summary>
        /// Provides an Icon for every component that will be visible in the User Interface.
        /// Icons need to be 24x24 pixels.
        /// </summary>
        protected override System.Drawing.Bitmap Icon
        {
            get
            {
                // You can add image files to your project resources and access them like this:
                //return Resources.IconForThisComponent;
                return null;
            }                                                コンポーネントのアイコン表示を定義
        }

        /// <summary>
        /// Each component must have a unique Guid to identify it.
        /// It is vital this Guid doesn't change otherwise old ghx files
        /// that use the old ID will partially fail during loading.
        /// </summary>
        public override Guid ComponentGuid
        {
            get { return new Guid("460cb157-91ba-4435-9417-ac5fe2628245"); }
        }
    }
}
```

作成するコンポーネントのアルゴリズムを確認

まず作成するコンポーネントについて説明する。はじめに示したように、作成するのは 2 点間を 1 周期とする sin 曲線をアウトプットするコンポーネントである。2 点は X 軸上にあり、sin 波は Z 軸方向に振動するものとしておく。

Grasshopper によるビジュアルコーディングでは、例えば以下のようなコードになる。



このコードについて理解する必要はないが、生成アルゴリズムを含むコンポーネントの処理の流れを理解することは必要である。コンポーネント全体の流れとしては以下のようになるだろう。

①始点および終点の X 座標を入力する
②二点間を分割数(10 とする)で分割した点を得る
③それぞれの分割点 X 座標について sin を計算し、Z 方向移動量とする
④分割点を Z 方向移動量分だけ移動させる
⑤移動後の各点を結んで sin 曲線とする
⑥sin 曲線を出力する

以降、testComponent.cs の流れに沿って以上の①から⑥のそれぞれを実装していく。

コンポーネント名やカテゴリーの定義

ここではははじめに設定したコンポーネントの表示名、説明、カテゴリーが”コンポーネント名”、”ニックネーム”、”コンポーネント説明”、”カテゴリー名”、”サブカテゴリー名”の順に示されている。ここで特に変更することはない。

```
public testComponent()
  : base("name", "nickname",
      "Construct an Archimedean, or arithmetic, spiral given its radii and number of turns.",
      "category", "subcategory")
```

コンポーネントのインプットを定義

ここではコンポーネントのインプットが定義されている。インプットは

pManager.インプットの型(名前、表示名、説明、データアクセスのタイプ、デフォルト値)

と定義される。ここでは以下のように２つのインプットを定義する。

```
protected override void RegisterInputParams(GH_Component.GH_InputParamManager pManager)
{
    pManager.AddNumberParameter("Start", "S", "X coordinate of Start Point", GH_ParamAccess.item, 0.0);
    pManager.AddNumberParameter("End", "E", "X coordinate of End Point", GH_ParamAccess.item, 10.0);
}
```

型	名前	表示名	説明	アクセスタイプ	デフォルト値
Number	Start	S	X coordinate of Start Point	GH_ParamAccess.item	0.0
Number	End	E	X coordinate of End Point	GH_ParamAccess.item	10.0

コンポーネントのアウトプットを定義

ここではインプットとほぼ同様にコンポーネントのアウトプットが定義されている。アウトプットは

pManager.インプットの型(名前、表示名、説明、データアクセスのタイプ)

と定義される。デフォルト値がないことに注意する。

```
protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager pManager)
{
    pManager.AddCurveParameter("Sin Curve", "S", "Output Sin Curve", GH_ParamAccess.item);
}
```

型	名前	表示名	説明	アクセスタイプ
Curve	Sin Curve	S	Output Sin Curve	GH_ParamAccess.item

コンポーネントの内部処理を定義

ここではコンポーネントの核となる内部処理を記述する。

記述するべき処理は

②二点間を分割数 10 で分割した点を得る

③それぞれの分割点 X 座標について sin を計算し、Z 方向移動量とする

④分割点を Z 方向移動量分だけ移動させる

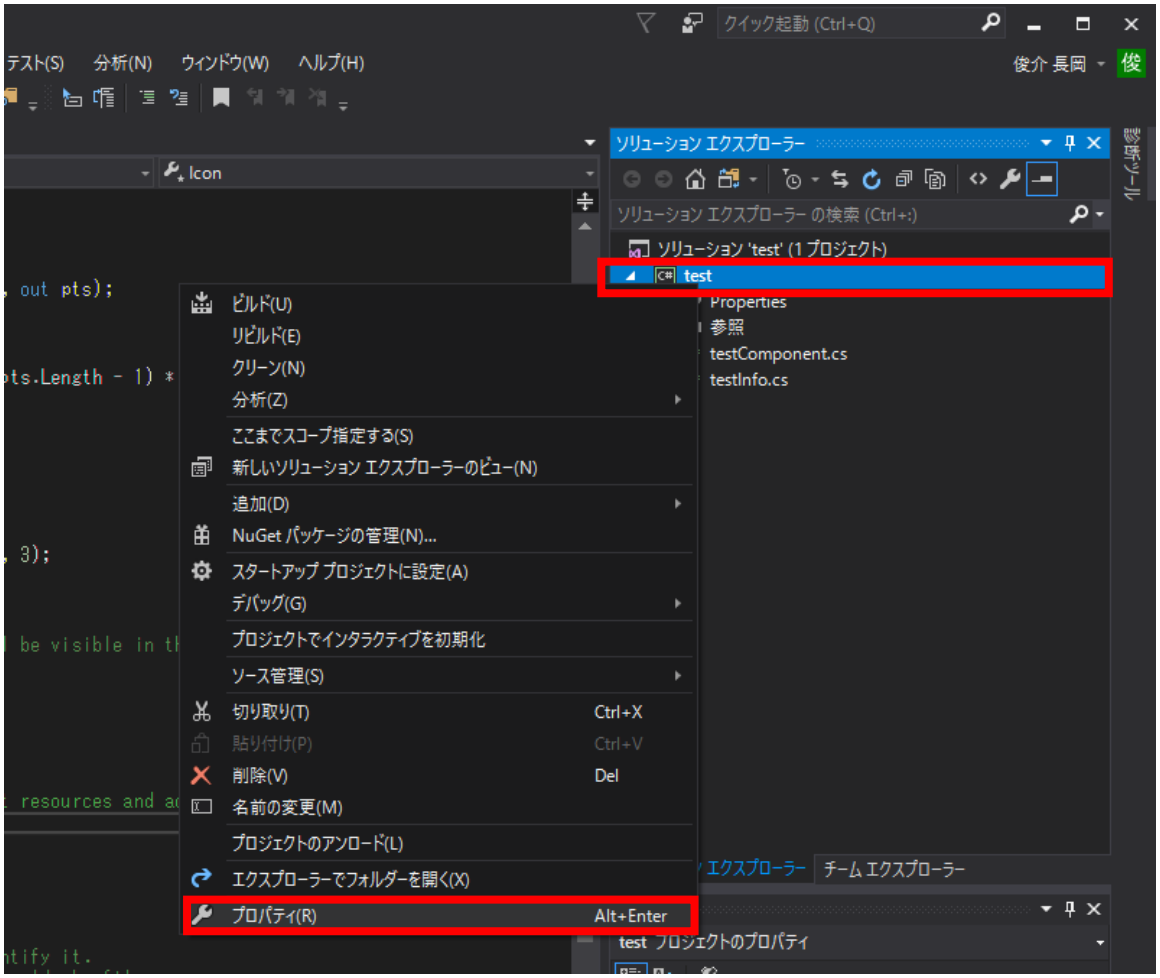
⑤移動後の各点を結んで sin 曲線とする

であり、以下のように書ける。

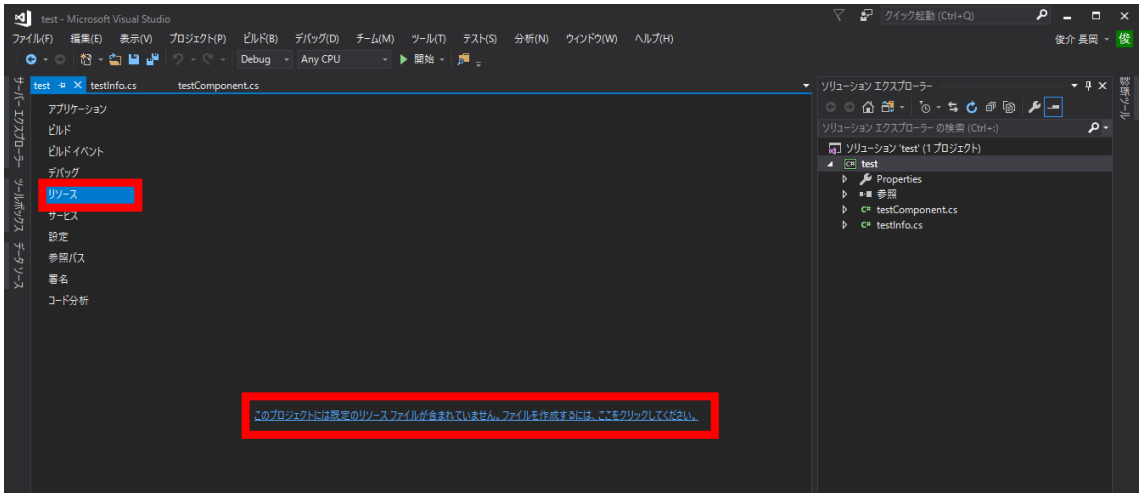
```
protected override void SolveInstance(IGH_DataAccess DA)
{
    double start = 0.0;
    double end = 10.0;
    int count = 10;

    if (!DA.GetData(0, ref start)) return;        インプットを定義時のインデックスで読み込み
    if (!DA.GetData(1, ref end)) return;

    if (start > end)
    {
        AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "E must be bigger than S");
        return;
    }                                                          例外処理の記述

    Curve Sin = CreateSinCurve(start, end, count);

    DA.SetData(0, Sin);                           アウトプットを定義時のインデックスで書き出し
}

private Curve CreateSinCurve(double start, double end, Int32 count)
{
    Point3d startPt = new Point3d(start, 0, 0);
    Point3d endPt = new Point3d(end, 0, 0);

    Line line = new Line(startPt, endPt);

    Point3d[] pts;
    line.ToNurbsCurve().DivideByCount(count, true, out pts);

    for(int i=0; i<pts.Length; i++)
    {
        double z = Math.Sin(i / Convert.ToDouble(pts.Length - 1) * 2* Math.PI);
        Point3d pt = pts[i];
        Point point = new Point(pt);
        point.Translate(new Vector3d(0, 0, z));

        pts[i] = point.Location;
    }

    Curve Sin = Curve.CreateInterpolatedCurve(pts, 3);
    return Sin;
}
```

コンポーネントのアイコン表示を定義

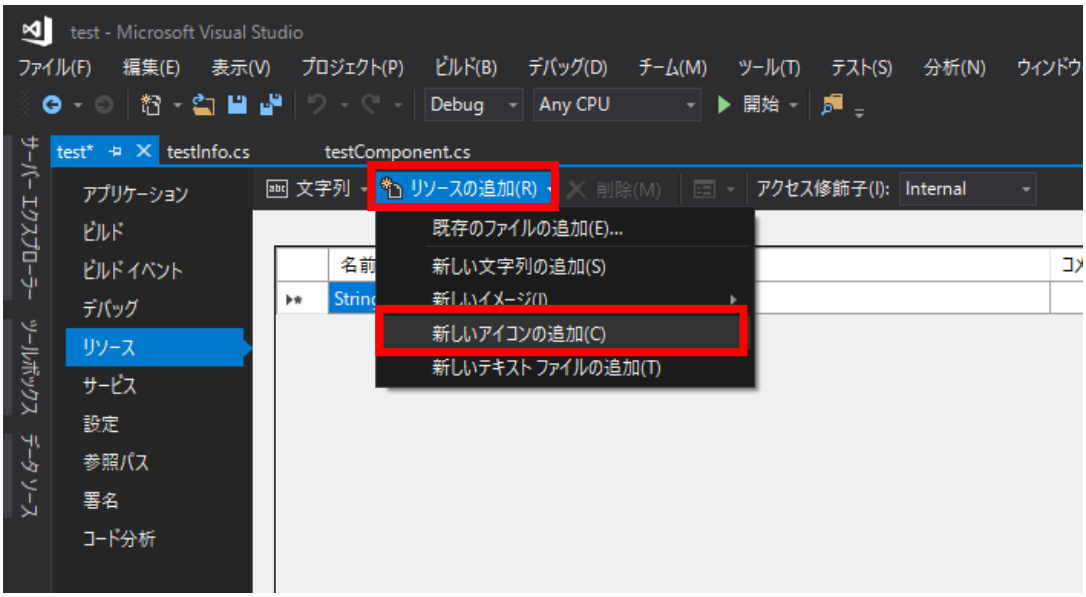
ここではコンポーネントとして書き出された際のアイコンを設定する。
はじめにアイコンを作成する。
ソリューションエクスプローラーのプロジェクト名(ここでは test)を右クリックし、プロパティをクリックする。



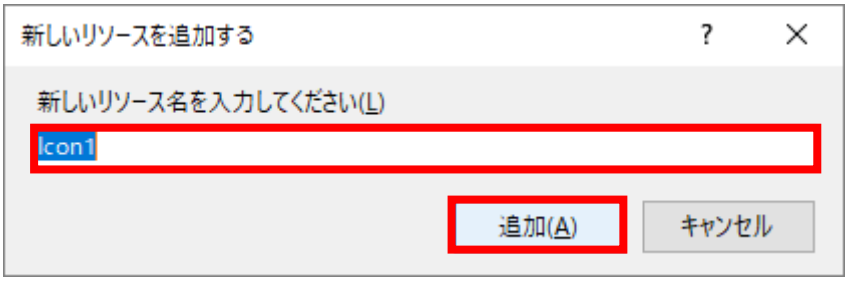
プロパティウィドウが表示されるので、左側のタブからリソースを選択する。リソースファイルが存在しない旨の表示がされるので、表示文をクリックしてリソースファイルを作成する。



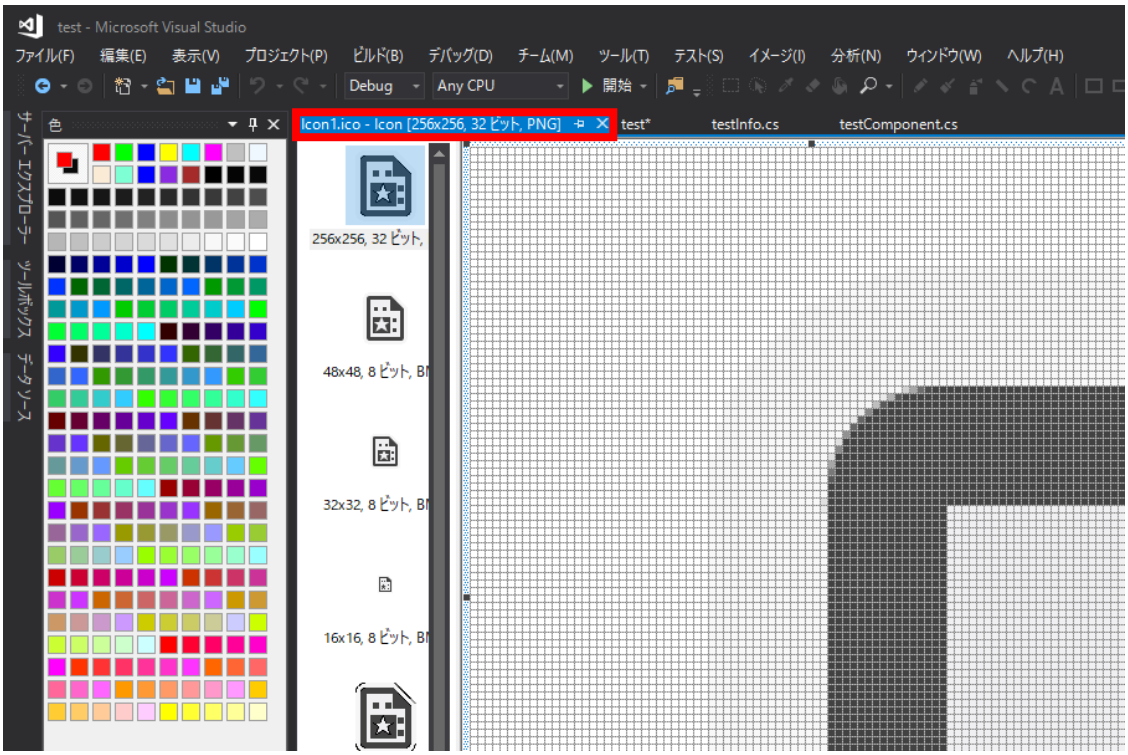
リソーステーブルが表示されるので、リソースの追加＞新しいアイコンの追加をクリックする。



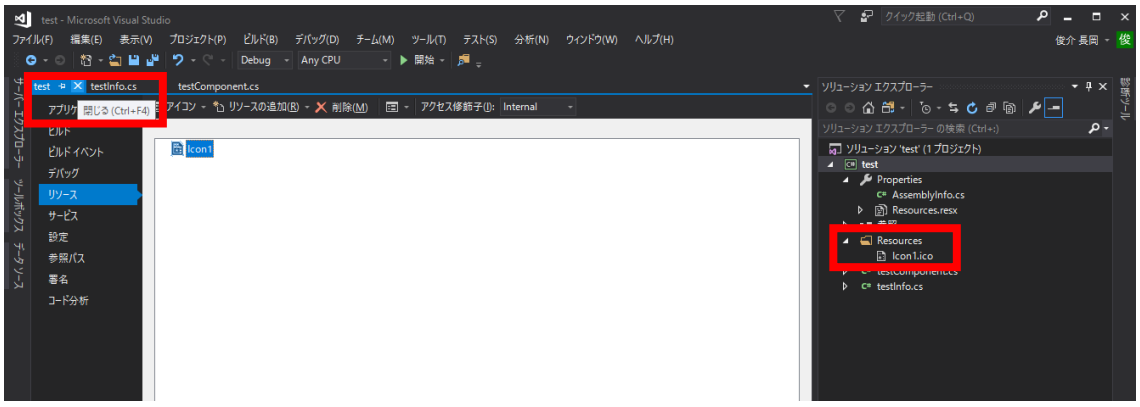
リソース(アイコン)名を入力して追加をクリックする。
ここでは既定の Icon1 としている。



新規アイコンのエディターが表示され、アイコンを編集することができる。ここでは表示されたままのアイコンを使用する。アイコンを保存し、×をクリックしてアイコンを閉じ、プロジェクトを保存する。



プロジェクトのプロパティウィンドウもプロジェクトの保存を行った後に×をクリックして閉じる。



右側のソリューションウィンドウに作成したアイコンが表示されていることを確認する。

アイコンファイルを作成することができたので、もとの testComponent.cs にアイコン設定を記述する。アイコン設定は

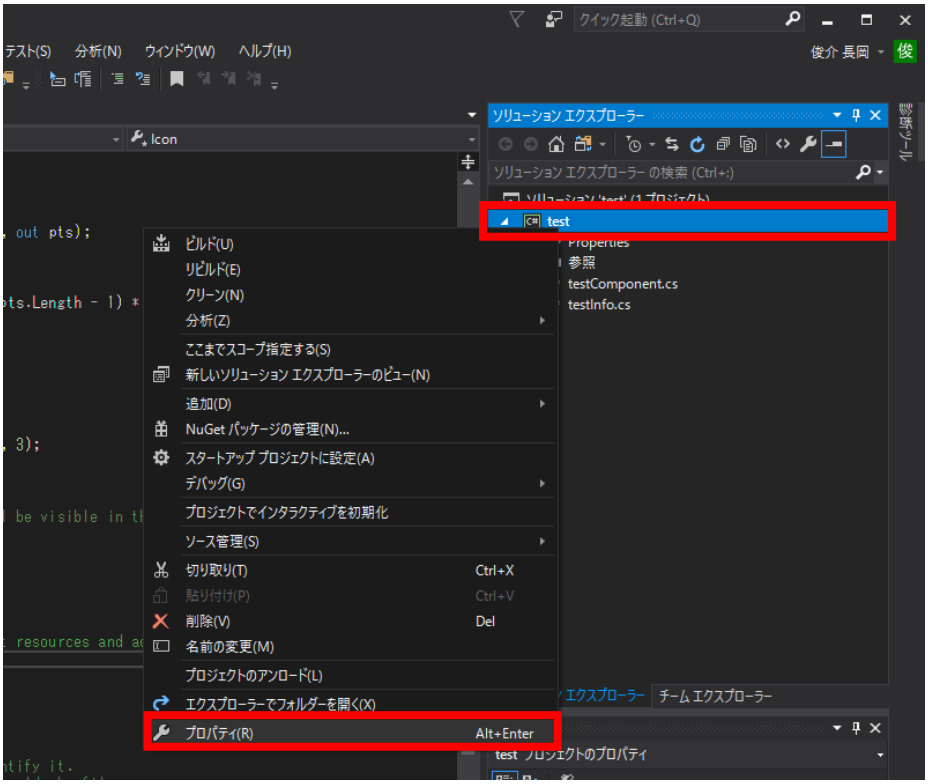
プロジェクト名.Properties.Resources.アイコン名.ToBitmap();

と定義される。

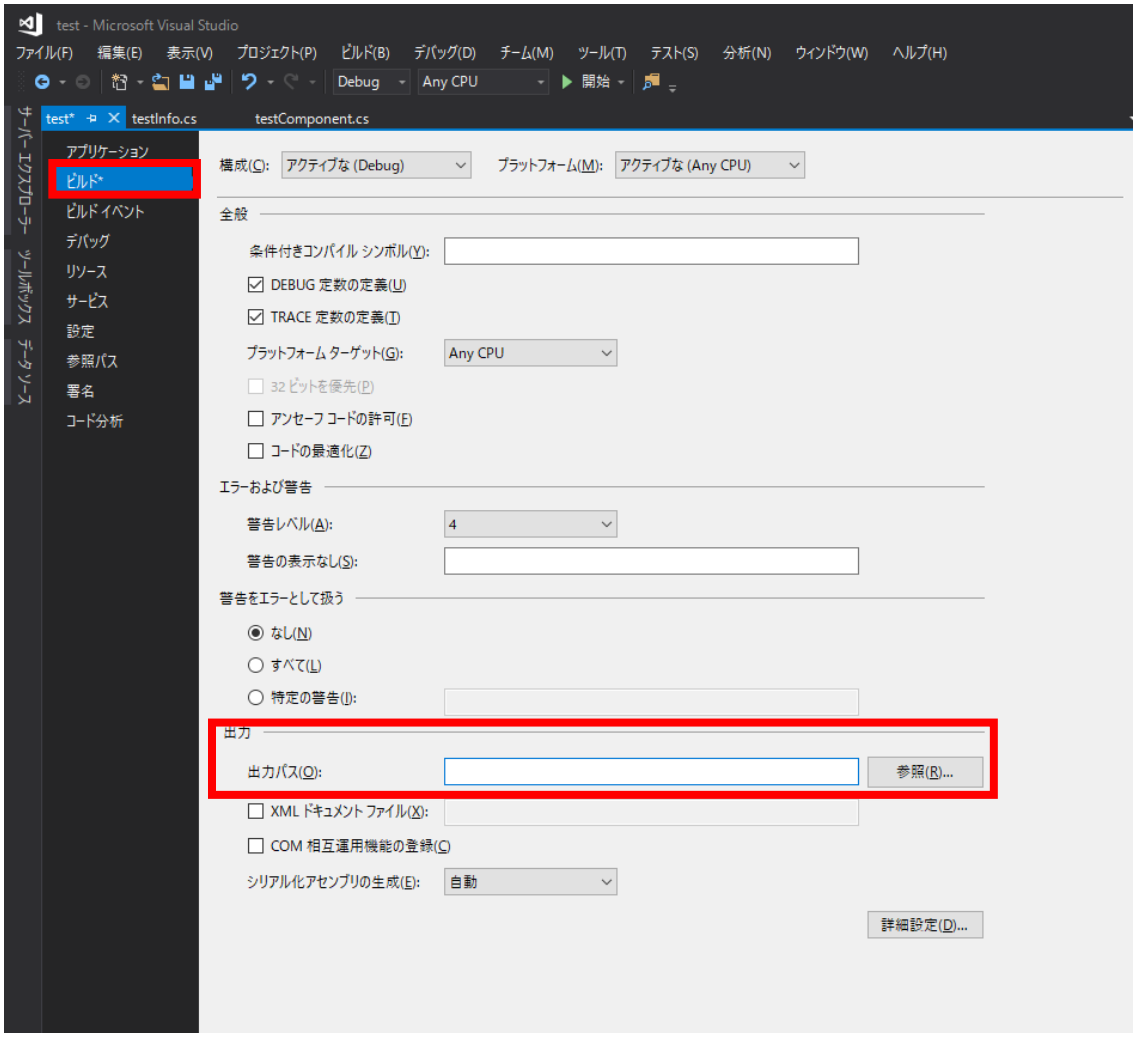
```
protected override System.Drawing.Bitmap Icon
{
    get
    {
        // You can add image files to your project resources and access them like this:
        //return Resources.IconForThisComponent;
        return test.Properties.Resources.Icon1.ToBitmap();
    }
}
```

.gha アウトプット先の設定

以上で必要な作業は終了するが、最後に.gha(grasshopper assembly file)ファイルの書き出し先を変更しておく。規定の設定では作成された.gha ファイルは作業ディレクトリに書き出されるが、一々Grasshopper のコンポーネントディレクトリに移すのは手間であるので、書き出し先をコンポーネントディレクトリに変更しておくとよい。アイコン作成時と同じようにプロジェクト名を右クリックし、プロパティをクリックする。

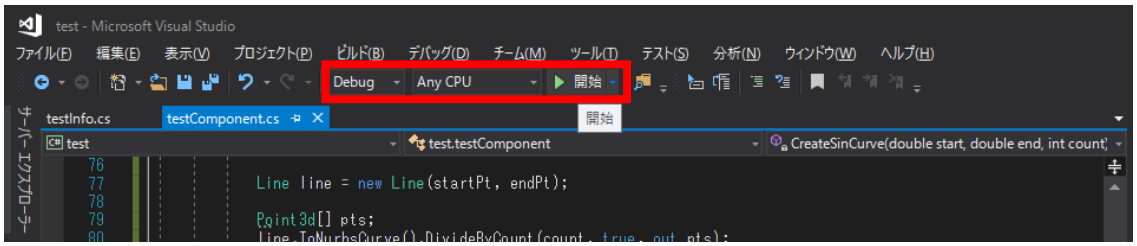


左側のタブからビルドを選択し、出力パスに Grasshopper のコンポーネントディレクトリのパスを記述する。これで出力パスの変更ができたので、保存してプロパティを閉じる。

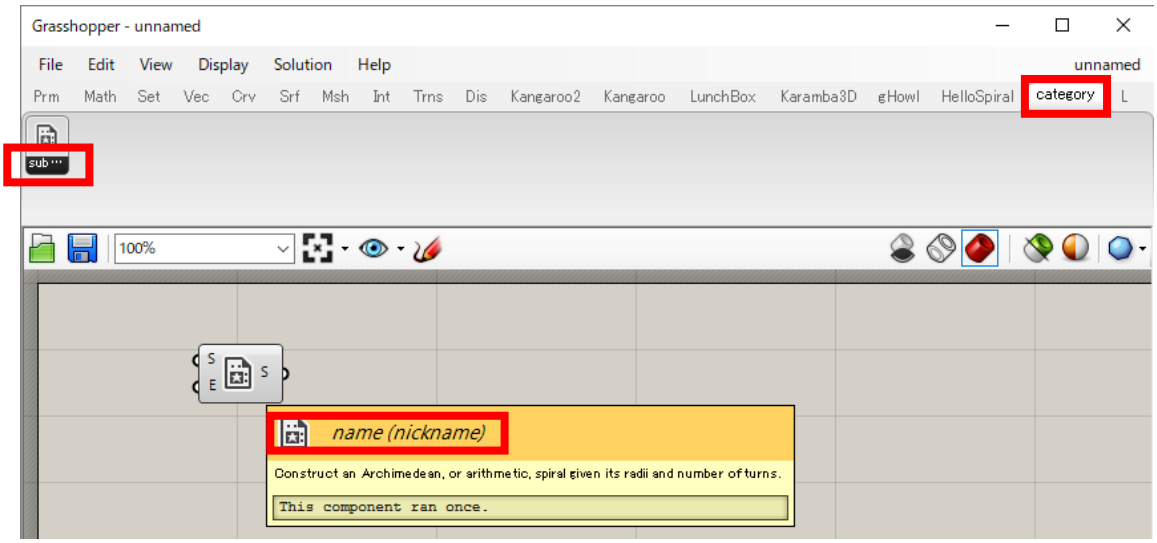


ソリューションのデバッグ

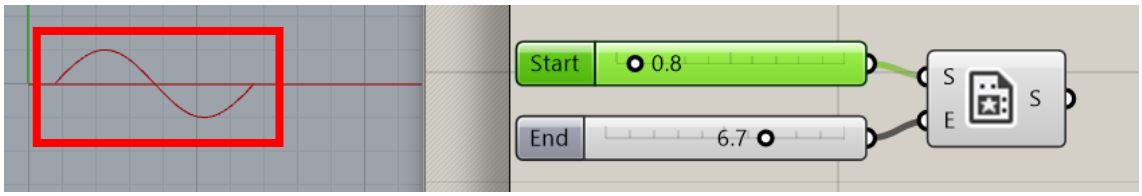
作成したソリューションをデバッグし、.gha ファイルを作成する。
開始をクリックするとデバッグが始まり、問題がなければ VS によって自動的に Rhinoceros が起動する。



その後 Grasshopper を起動すると category＞subcategory＞name コンポーネントが作成されている。これは先ほどの設定によって既に Grasshopper のコンポーネントディレクトリに.gha ファイルが生成されているためである。



name コンポーネントを配置し、想定通りに sin 曲線が生成されることを確認する。



End より Start のほうが大きいと、エラーメッセージが表示される。