C#で逝く AtsEX 入門

T.Wattz

2024 年 6 月 13 日

こんにちは。T.Wattz です。
早速ですが、AtsEX とはなにかについて少し説明をしましょう。

What is AtsEX??

鉄道シミュレータ「BVE Trainsim」のプラグイン（Ats プラグイン）を拡張（ex tend）する C#ライブラリ。
DirectX にまで干渉できるほど拡張性は高く、知識と技術さえあれば基本的に何でも作れてしまうスゴい代物。
文化祭で公開したゲーム「MetroDrive 日比谷編」も本体にはこれを使用している。

お察しかとは思いますが、そこそこの程度の C#の知識は必要です。（Unity が満足に使えるレベルなら十分）
あとは、要所要所で Unity の類似機能を紹介しながらなので、Unity 勢にはわかりやすいかもしれません。
BVE は Windows ソフトなので Mac では動かない点はご注意。（プラグインを使うと Wine を使っても）

1. 主要機能の説明

まずは主要機能について軽く説明します。
（読み飛ばして頂いても構いません。）

PluginMain

PluginMain は Unity でいうところの Start 関数（厳密には違う）です。
ここで最初に一回呼び出したいものやイベントの登録などを行います。

リスト 1　Sample

```
1 public MapPluginMain(PluginBuilder
      builder) : base(builder)
```

と宣言することで使用可能になります。

Dispose

シナリオの終了時に呼び出す関数です。(Unity で言えば OnApplicationQuit() が近い.... かも)
イベントの購読解除とかに使います。

TickResult

Unity の Update 関数にあたるもので、シナリオが読み込まれた後毎フレーム呼び出されるものです。
詳細は後述しますが、BveHacker.Scenario などシナリオの実行中にしか読めない値はここでしか読めません。
基本的にはフレーム間で変わらないかもしれないもの（速度など）は毎フレーム代入するスタンスで OK です。

2. プラグインを作ろう

では、実際にプラグインを制作していきましょう。
今回の最終的な目標は「DirectX で力行ノッチの値、ブレーキノッチの値を表示する」こととします。

環境設定

github.com/stop-pattern/AtsExCsTemplate を code → Download zip より Zip で落とし、任意の場所に展開しましょう。
次に、Visual Stadio をインストールします。（ここらへんはアップデートのたびにやり方が変わりそうなので、適当にググってください）
ワークロードは「.NET デスクトップ開発」をインストールすれば OK です。
インストールが済んだら MapPlugin\AtsExCsTemplate.sln を開いてください。
開いたら検索/nuget パッケージの管理 より、「プレリリースを含める」にチェックを付け、参照¿AtsEX と検索して AtsEx.CoreExtions と AtsEx.PluginHost をインストールしてください。
AtsEX は執筆中の現在 (2024-05-07 時点) あくまで正式リリース**候補**なので、最新バージョンかどうかは https://www.okaoka-depot.com/AtsEX/download/ からご確認ください。（今時点では

ver1.00-RC8 が最新バージョンです。)
では MapPlugin.cs を編集していきましょう。まずは最低限のコードでビルドできるか確認してみます。

リスト 2　MapPlugin.cs

```
using System;
using AtsEx.PluginHost.Plugins;
namespace BuildSample
{
/// プラグインの本体
[PluginType(PluginType.MapPlugin)]
internal class MapPluginMain :
      AssemblyPluginBase
{
    int sample;
    bool isFive;
    /// プラグインが読み込まれた時に呼ばれる
    /// 初期化を実装する
    public MapPluginMain(PluginBuilder
          builder) : base(builder)
    {
      sample = 1
      isFive = false;
    }
    /// プラグインが解放されたときに呼ばれる
    /// 後処理を実装する
    public override void Dispose()
    {}
    /// シナリオ読み込み中に毎フレーム呼び出
          される
    public override TickResult Tick(
          TimeSpan elapsed)
    {
      sample++;
      if(sample == 5)
      {
        isFive = true;
      }
      return new MapPluginTickResult();
    }
  }
}
```

以上のコードは特に意味がないものなのですが、一応これをベースとしてやっていきます。
これをコピペしても多分エラーは出ないと思いますが、もし出たら後述の「トラブルシューティング」をご確認ください。
では、ビルドしていきましょう。といっても、やることは簡単で ctrl+B を押すだけです。
コードのウインドウの左下に「問題は見つかりませんでした」と表示されていれば成功するはずです。
するとカレントディレクトリ\bin\Debug\net48 の中にプラグインの dll が生成されたと思います。
もし指定のディレクトリの中になければ、「プロジェクト」「デバッグ」と書いてある箇所の下を「debug」「AnyCPU」と変更します。
エラーが出てビルドに失敗する場合は、AtsEx.PluginHost がインストールされていない可能性があります。もう一度 nuget を確認してください。
ビルドが完了したら以下の状態に戻してください。

リスト 3　MapPlugin.cs

```
using System;
using AtsEx.PluginHost.Plugins;
namespace BuildSample
{
  [PluginType(PluginType.MapPlugin)]
  internal class MapPluginMain :
        AssemblyPluginBase
  {
    public MapPluginMain(PluginBuilder
          builder) : base(builder)
    {}
    public override void Dispose()
    {}
    public override TickResult Tick(
          TimeSpan elapsed)
    {
      return new MapPluginTickResult();
    }
  }
}
```

では、実際にコードを編集していきます。
今回のゴールは力行（マスコン）/ブレーキの値の表示なので、まずはそれらの値を取得するところから始めましょう。
結論から言うと、

リスト 4　MapPlugin.cs

```
using System;
using AtsEx.PluginHost.Plugins;

namespace DrawNotch;
{
  [PluginType(PluginType.MapPlugin)]
  internal class MapPluginMain :
        AssemblyPluginBase
  {
    int power;
    int break;
    public MapPluginMain(PluginBuilder
          builder) : base(builder)
    {}
    public override TickResult Tick(
          TimeSpan elapsed)
    {
      power = Native.Handles.Power.
            Notch;
      break = Native.Handles.Break.
            Notch;
      return new MapPluginTickResult();
    }
  }
}
```

となるのですが、おそらく実際にはこの値のみを使ってプラグインを作ることは少ないと思うので、その他の機能についても簡潔に触れておこうかと思います。
(といっても、機能が多すぎるために特に分かりづらいもののみですが...)　他の機能についてはオブジェクトブラウザーを使うのがいいのですが、やっぱり公式の Discord 鯖で質問しないとわからないことも多々あります (筆者のプログラミング能力が低いだけかも)
okaoka-depot.com/AtsEX/wiki/
こちらからどうぞ。
~~質問しすぎるとシカトされます~~

リスト 5　機能リスト

```
//BveTypes.ClassWrappers\
      Station に入っている値(通過時刻とか)
      を参照する場合
var station = BveHacker.Scenario.Route.
      Stations[(調べたい駅のインデックス、変
      数も可能)] as Station;
if (station == null){
    arriveMilli = station.
          ArrivalTimeMilliseconds;//通過
          時刻 (ミリ秒)
    passMilli = station.
          DepartureTimeMilliseconds;//停
          車時刻 (ミリ秒)
    pass = station.Pass;//通過/停車の判定
}
//駅インデックス
index = BveHacker.Scenario.Route.
      Stations.CurrentIndex + 1;//停車時
      は前駅のインデックスが代入され、開始時
      に
      index がマイナスになることを避けるために
      +1している
//地上子通過イベントの登録 (PluginMain 内)
Native.BeaconPassed += new
      BeaconPassedEventHandler((呼び出し
      たい関数名));
//地上子通過イベントの購読解除 (Dispose 内)
Native.BeaconPassed -= new
      BeaconPassedEventHandler((呼び出し
      たい関数名));
//地上子イベント発生時に呼び出す関数
public void BeaconPassed(
      BeaconPassedEventArgs e)//e.
      Type で地上子のタイプを指定できる (読み出し)

{
```

```
17      switch (e.Type)//メトロ総合プラグインの場合
18      {
19          case 10://信号 0
20              atc = 0;
21              break;
22          case 19://ATC45
23              atc = 45;
24              break;
25      }
26    //他にも e.SignalIndex(int)で対となる閉塞のインデックス、e.Distance(float)で対となる閉塞までの距離を取得可能。
27  }
```

こんなもんでしょうか（長くてすみません）

正直 AtsEX はできることが多すぎる上、Unity ほど文献が充実していないので、繰り返になりますが公式 Discord の利用がおすすめです（ちなみに、設立は**俺**が提案しました。えらい。）

余談にはなりますが、個人的に Unity が使いやすいのはそういう所だったりします。Godot や Unreal などは文献的な面で Unity 程初心者が色々できるような代物ではないのかな〜と思います。(知らんけど)

では開発の方に戻りましょう。

以上では力行ノッチとブレーキノッチの値を取得しました。今度は、それが実行できているかを確認してみましょう。

以下のコードを PluginMain 内に記述してください。

リスト 6　MapPluginMain()

```
1  if (!System.Diagnostics.Debugger.IsAttached)
2  {System.Diagnostics.Debugger.Launch();}
```

これはデバッガーを起動するためのもので、VisualStadio がインストールされてる場合は自動的にデバッガーが起動します。

勿論 Plugin だけでは動作しないので、BveTrainism5.8(と DirectX9.0cRunTime,.NET Framework3.5),AtsEX をインストールしましょう。（ここはあまり関係ないので省略）

次に車両を用意します。（ここもググること前提で省略します。）

Map ファイルは以下のようにします。

リスト 7　Map.txt

```
1  BveTs Map 2.02
2  include '[[AtsEx::NOMPI]] このマップではAtsEX マッププラグインを使用しています。正常動作には AtsEX のインストールが必要です。詳細は https://automatic9045.github.io をご覧ください。';
3  include '[[AtsEx::NOMPI]] This map requires to install AtsEX to execute map plugins. For more information, visit https://automatic9045.github.io';
4  include '<AtsEx::USEATSEX>';
5  include '<AtsEx::READDEPTH>1';
6  include '<AtsEx::MapPluginUsing>MapPluginUsing.xml';
```

（ご自身の Map に組み込む場合は上 4 行をメインの Map ファイルの最上段に組み込んで下さい）

次に MapPluginUsing.xml を作成して Map.txt と同ディレクトリに置きます。

内容は以下の通りです。

リスト 8　MapPluginUsing.xml

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <AtsExPluginUsing xmlns="http://automatic9045.github.io/ns/xmlschemas/AtsExPluginUsingXmlSchema.xsd">
3    <Assembly Path="(プラグインdll へのパス)" />
4  </AtsExPluginUsing>
```

これで動くはずです。（動かない場合は、パスを確認して下さい。）

次に、ブレークポイントの設定をします。

MapPlugin.cs に戻って下さい。そして、コードの左側をクリックして power = Native.Handles.Power.Notch; の左にブレークポイントを設定して下さい。

では、実際に BVE を起動し、シナリオを読み込んで下さい。（上に同じく）

すると、以下画像のようにデバッガーが起動すると思います。

起動したら、以下画像のようになるかと思います。

ブレークポイントで止まると思いますので、止まったら（BveTs.exe が応答しなくなったら）VisualStadio のウインドウに移動し、左下を御覧ください。

すると、おそらく現在の power,break の値が表示されると思います。（通常のシナリオなら、power は 0,break は非常ブレーキの値に）では、早速これを反映する UI を作成していきましょう。

まずは DirectX を最低限触ります。Nuget より Lib.Harmony,ObjectiveHarmonyPatch,SlimDX をインストールし、using ディレクティブには FastMember,TypeWrapping,ObjectiveHarmonyPatch を追加して下さい。

リスト 9　MapPluginMain()

```
1  ClassMemberSet assistantDrawerMembers = BveHacker.BveTypes.GetClassInfoOf<AssistantDrawer>();
2  FastMethod drawMethod = assistantDrawerMembers.GetSourceMethodOf(nameof(AssistantDrawer.Draw));
3  HarmonyPatch drawPatch = HarmonyPatch.Patch(Name, drawMethod.Source, PatchType.Prefix);
4  drawPatch.Invoked += (sender, e) =>
5  {
6    //ここに実際の処理を書く
7  };
```

このコードは、BVE 本体にパッチを当てるもので、BveHacker.BveTypes.GetClassInfoOf<AssistantDrawer>() では BVE 本体の補助表示のコードに対するものです。

つまり、これを唱えれば補助表示の位置 (運転台よりも手前) に描画することが可能という訳です。

実際の処理を追加していきましょ

う。たいして冗長なコードでもないですが、練習も兼ねてクラス(ファイル) を分けてみましょう。クラス名を Drawer として解説します。

リスト 10　クラス直下〜PluginMain

```
internal class MapPluginMain : AssemblyPluginBase
{
  Drawer drawer;\\クラスを変数として宣言する
  public MapPluginMain(PluginBuilder builder) : base(builder)
  {
    drawer = new Drawer();//新たにDrawer の処理を呼び出す
    ~(以下略)~
  }
}
```

というわけで、これだけで使えます。簡単。自分も初めは誤解していたのですが、new クラス名 () はあくまで「初期化」であって「スクリプトを使用する許可」ではない点にご注意。これを間違えて 2 回以上呼び出すと変数の値が呼び出した場所ごとに違ってごちゃごちゃになります。

次に、Drawer.cs を書きましょう。割と脳筋なので、改善点は結構あります。(TickResult で呼び出すものではないので, 配列を使うと最初の値のまま返歌しなくなります。) 紙面節約の観点のから power,break ともに 1 つずつしか書きませんが、合う分だけ書き足して下さい。

リスト 11　Drawer.cs

```
using AtsEx.PluginHost.Plugins;
using BveTypes.ClassWrappers;
using System.IO;
using System.Drawing;
using SlimDX;
using SlimDX.Direct3D9;
namespace DrawNotch()
{
  internal class Drawer()
  {
    Model p0;
    Model b0;
    public void CreateModel(string location)
    {
      p0 = CreateModels(@"picture\power\0.png",150,-225);//第一引数にはdll からみた画像のパスを描く
      b0 = CreateModels(@"picture\break\0.png",150,-225);
    Model CreateModels(string path, float width,float height)
    {
      string texFilePath = Path.Combine(Path.GetDirectoryName(location), path);
      RectangleF rectangleF = new RectangleF(0, 0, width, height);
      Model model = Model.CreateRectangleWithTexture(rectangleF, 0, 0, texFilePath);//四角形の 3D モデル
      return model;
    }
  }
  public void Draw(int power,int break)
  {
    Device device = Direct3DProvider.Instance.Device;
    int width = Direct3DProvider.Instance.PresentParameters.BackBufferWidth;//ウインドウのwidth
    int height = Direct3DProvider.Instance.PresentParameters.BackBufferHeight;//ウインドウのHeight
    device.SetTransform(TransformState.World, Matrix.Translation(-width / 4, -height / 4, 0));//基準となる位置
    if(power==0){p0.Draw(Direct3DProvider.Instance, false);}//他のノッチ数も同様に
    device.SetTransform(TransformState.World, Matrix.Translation(-width / 4+40, -height / 4+40, 0));//位置は 2回以上記述可能
    if(break==0){b0.Draw(Direct3DProvider.Instance, false);}
  }
 }
}
```

できました。CreateModel() は 3D モデルを作成し、Draw() は 3D モデルを描画します。では、これを PluginMain の方で呼び出しましょう。

リスト 12　MapPluginMain()

```
drawer = new Drawer();//新たにDrawer の処理を呼び出す
ClassMemberSet assistantDrawerMembers = BveHacker.BveTypes.GetClassInfoOf<AssistantDrawer>();
FastMethod drawMethod = assistantDrawerMembers.GetSourceMethodOf(nameof(AssistantDrawer.Draw));
HarmonyPatch drawPatch = HarmonyPatch.Patch(Name, drawMethod.Source, PatchType.Prefix);
drawer.CreateModel()
drawPatch.Invoked += (sender, e) =>
{
  drawer.Draw(power,break);//リスト 4参考
};
```

これで完成です。お疲れ様でした。スクリプトの全体を書く前に、外部クラスでの TickResult の呼び出し方について触れておきます。

```
public TickResult 関数名 ()
{//処理}
```

これを Main の Tick 内で呼び出します。(int life のように引数を指定することも可能です。)
では最後にまとめて終わりましょう。

リスト 13　MapPluginMain.cs

```
using System;
using AtsEx.PluginHost.Plugins;
using FastMember;
using TypeWrapping;
using ObjectiveHarmonyPatch;

namespace DrawNotch;
{
  [PluginType(PluginType.MapPlugin)]
  internal class MapPluginMain : AssemblyPluginBase
  {
    int power;
    int break;
    public MapPluginMain(PluginBuilder builder) : base(builder)
    {
      drawer = new Drawer();
      ClassMemberSet assistantDrawerMembers = BveHacker.BveTypes.GetClassInfoOf<AssistantDrawer>();
      FastMethod drawMethod = assistantDrawerMembers.GetSourceMethodOf(nameof(AssistantDrawer.Draw));
      HarmonyPatch drawPatch = HarmonyPatch.Patch(Name, drawMethod.Source, PatchType.Prefix);
      drawer.CreateModel(Location);//ここでの
```

```
            Location は Bvehacker の Location
21        drawPatch.Invoked += (sender, e
              ) =>
22        {
23          drawer.Draw(power,break);
24        };
25      }
26      public override TickResult Tick(
            TimeSpan elapsed)
27      {
28        power = Native.Handles.Power.
              Notch;
29        break = Native.Handles.Break.
              Notch;
30        return new MapPluginTickResult
              ();
31      }
32    }
33  }
```

次に Drawer.cs。

リスト 14　Drawer.cs

```
1   using AtsEx.PluginHost.Plugins;
2   using BveTypes.ClassWrappers;
3   using System.IO;
4   using System.Drawing;
5   using SlimDX;
6   using SlimDX.Direct3D9;
7   namespace DrawNotch()
8   {
9     internal class Drawer()
10    {
11      Model p0;
12      Model b0;
13      public void CreateModel(string
            location)
14      {
15        p0 = CreateModels(@"picture\
              power\0.png",150,-225);//
              第一引数には
              dll からみた画像のパスを描く
16        b0 = CreateModels(@"picture\
              break\0.png",150,-225);
17        Model CreateModels(string path,
              float width,float height)
18        {
19          string texFilePath = Path.
                Combine(Path.
                GetDirectoryName(
                location), path);
20          RectangleF rectangleF = new
                RectangleF(0, 0, width,
                height);
21          Model model = Model.
                CreateRectangleWithTexture
                (rectangleF, 0, 0,
                texFilePath);//四角形の 3
                D モデル
22          return model;
23        }
24      }
25      public void Draw(int power,int
            break)
26      {
27        Device device =
              Direct3DProvider.Instance.
              Device;
28        int width = Direct3DProvider.
              Instance.PresentParameters
              .BackBufferWidth;//ウインド
              ウのwidth
29        int height = Direct3DProvider.
              Instance.PresentParameters
              .BackBufferHeight;//ウイン
              ドウのHeight
30        device.SetTransform(
              TransformState.World,
              Matrix.Translation(-width
              / 4, -height / 4, 0));//基
              準となる位置
31        if(power==0){p0.Draw(
              Direct3DProvider.Instance,
              false);}//他のノッチ数も同様
              に
32        device.SetTransform(
              TransformState.World,
              Matrix.Translation(-width
              / 4+40, -height / 4+40,
              0));//位置は 2回以上記述可能
33        if(break==0){b0.Draw(
              Direct3DProvider.Instance,
              false);}
34      }
35    }
36  }
```

ほんへは以上です。

3. おまけ

~~おまけとはいったものの~~
~~書くことがあんまりない~~