Lydian シミュレータ

マニュアル

Jazz Soft, Inc.

1. 改訂履歴

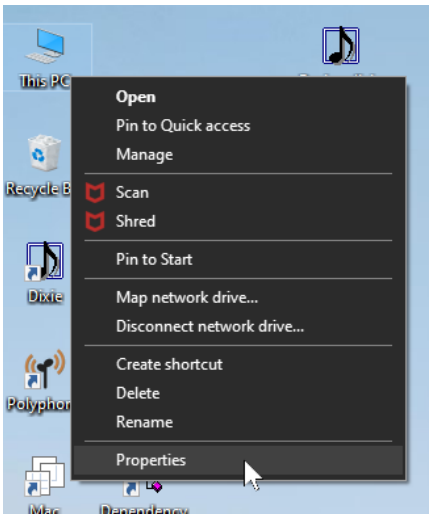
バージョン	日付	改訂者	説明
1.00	2021 年 12 月 5 日	Carl Okada	新規作成
1.00a	2021 年 12 月 7 日	Carl Okada	Dorian の PATH 設定を追加。
1.01	2021 年 12 月 7 日	Carl Okada	依存性の注入を変更。
1.02	2022 年 2 月 19 日	Carl Okada	自動返信を追加。

2. 目次

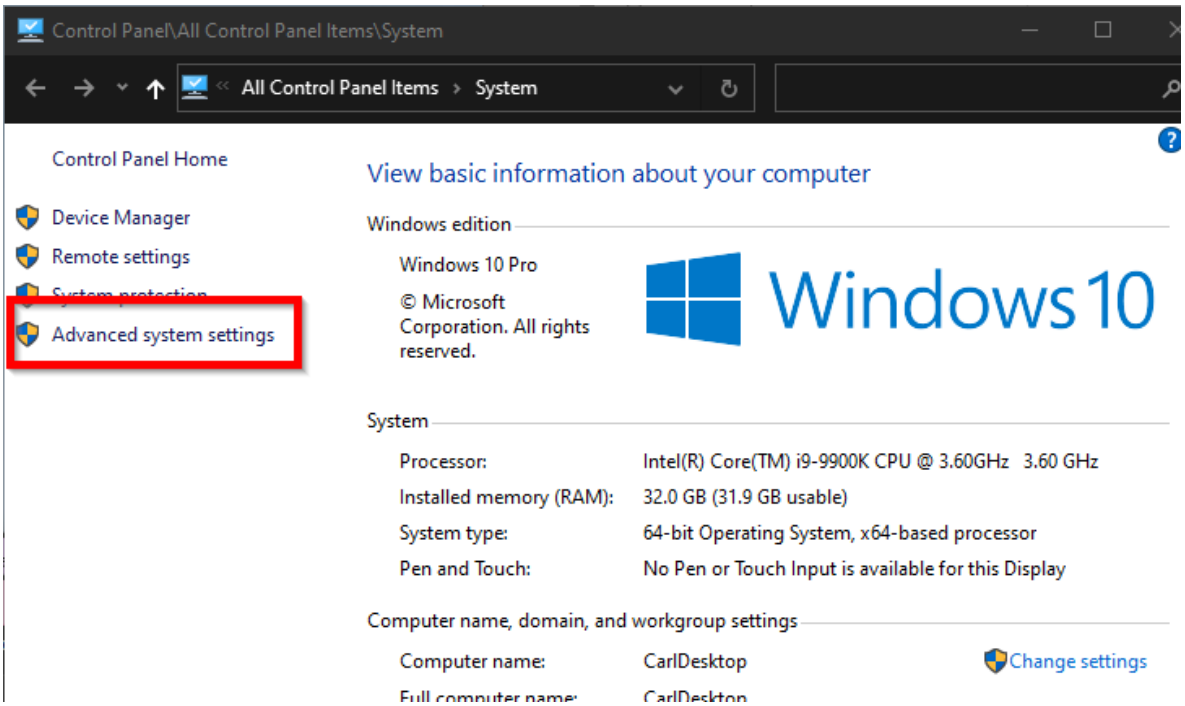
3. Dorian の設定

Dorian を使う場合は PATH への追加が必要となります。

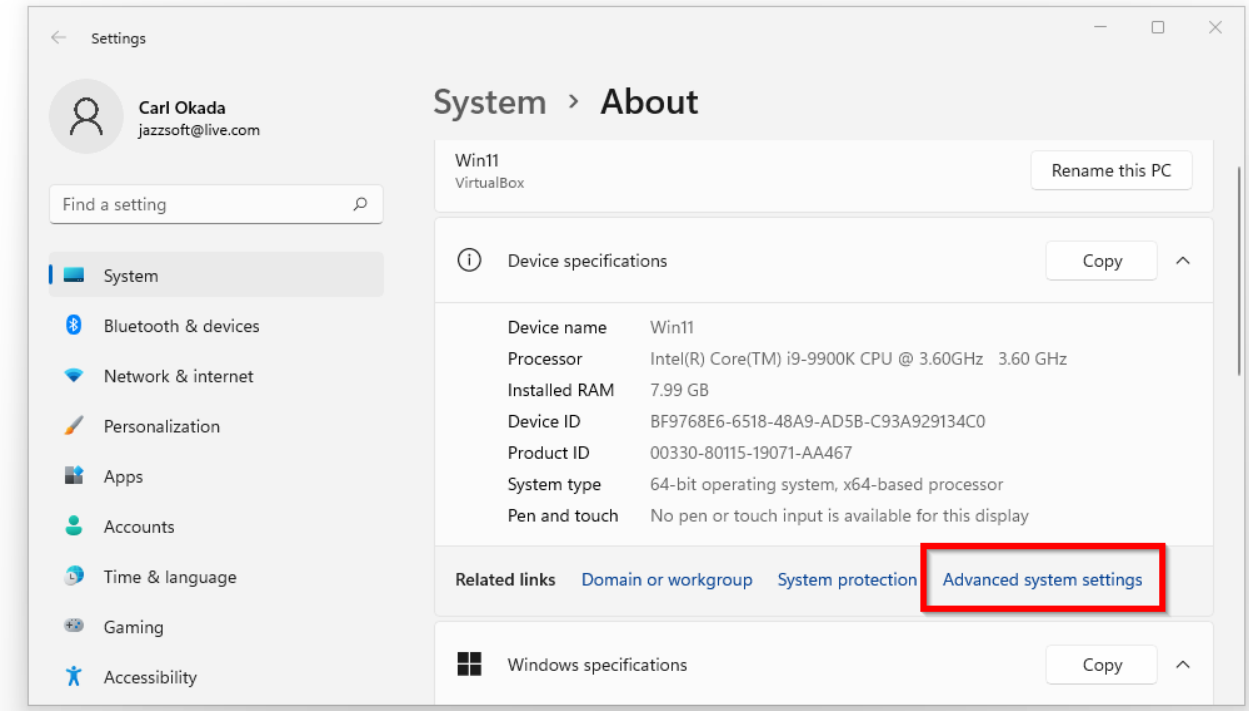
1 デスクトップにある「This PC」を右クリックし、プロパティをクリックします。



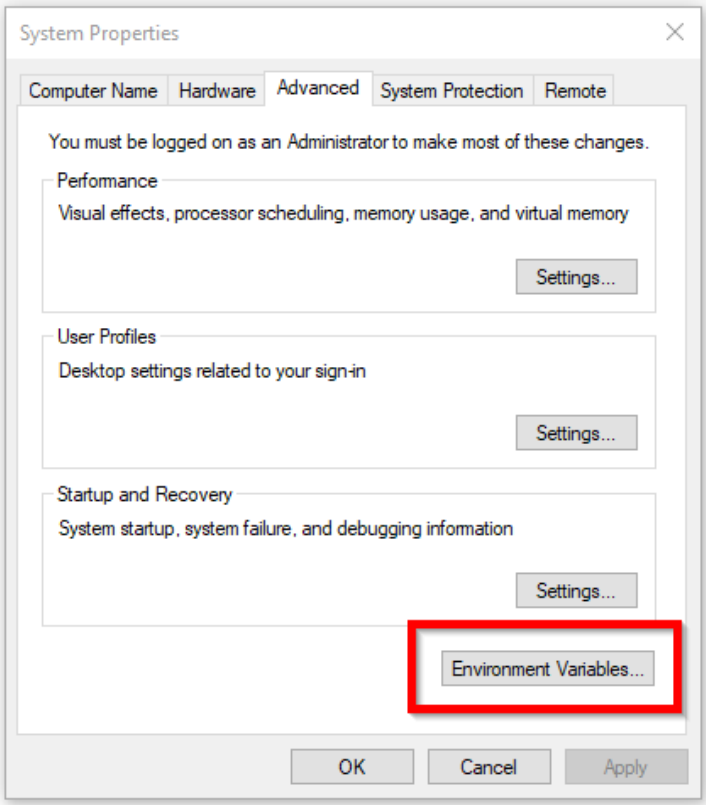
2 画面左側にある「Advanced system settings」をクリックします。



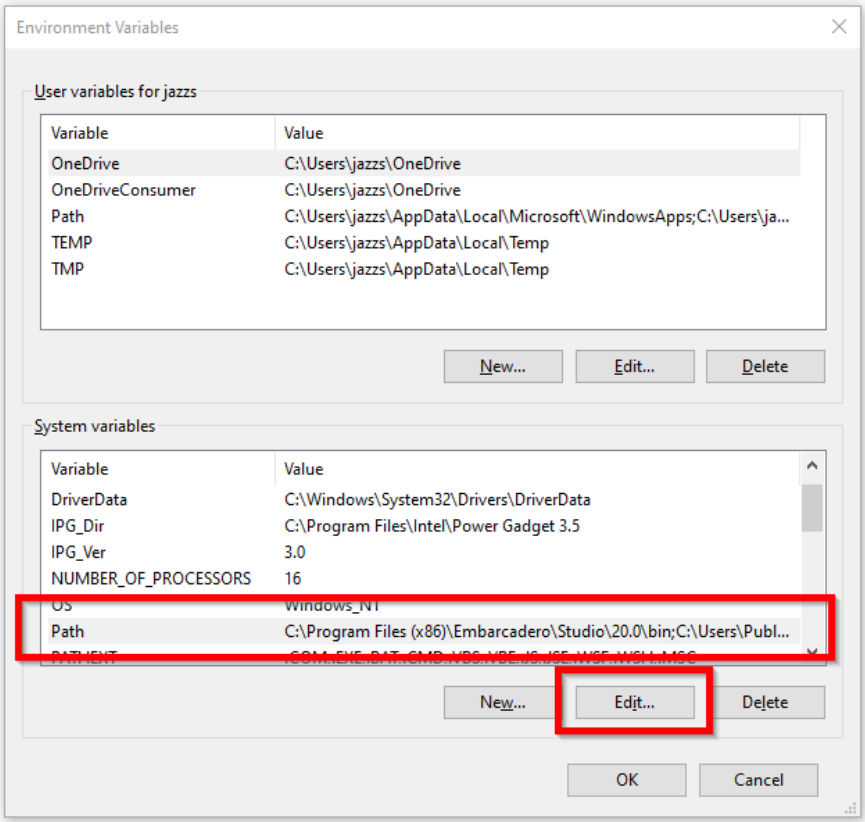
Windows 11 の場合は画面中央にあります。



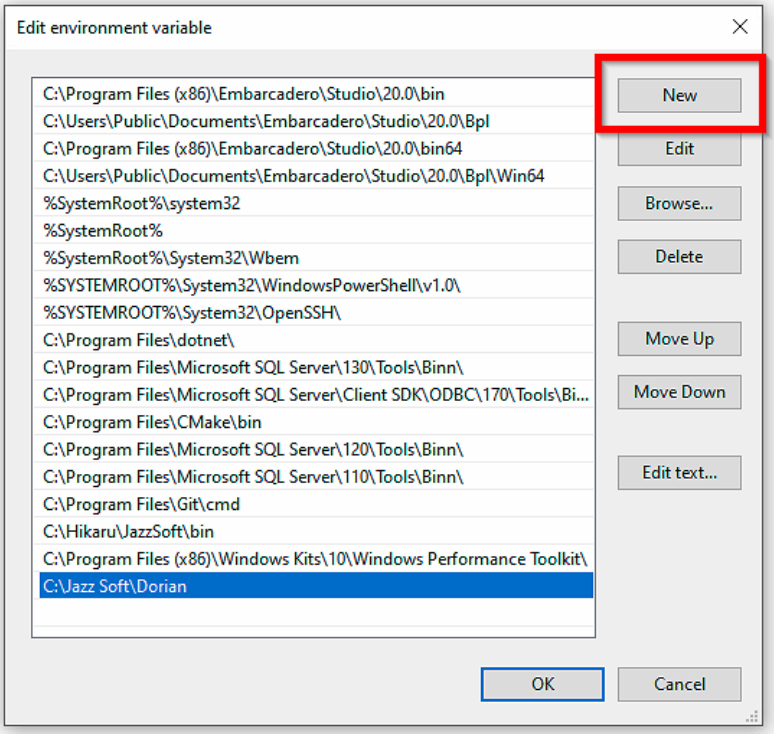
3 「Environment Variables...」をクリックします。



4 「System variables」（下側）の「Path」を選択し、その下にある「Edit...」ボタンを押します。

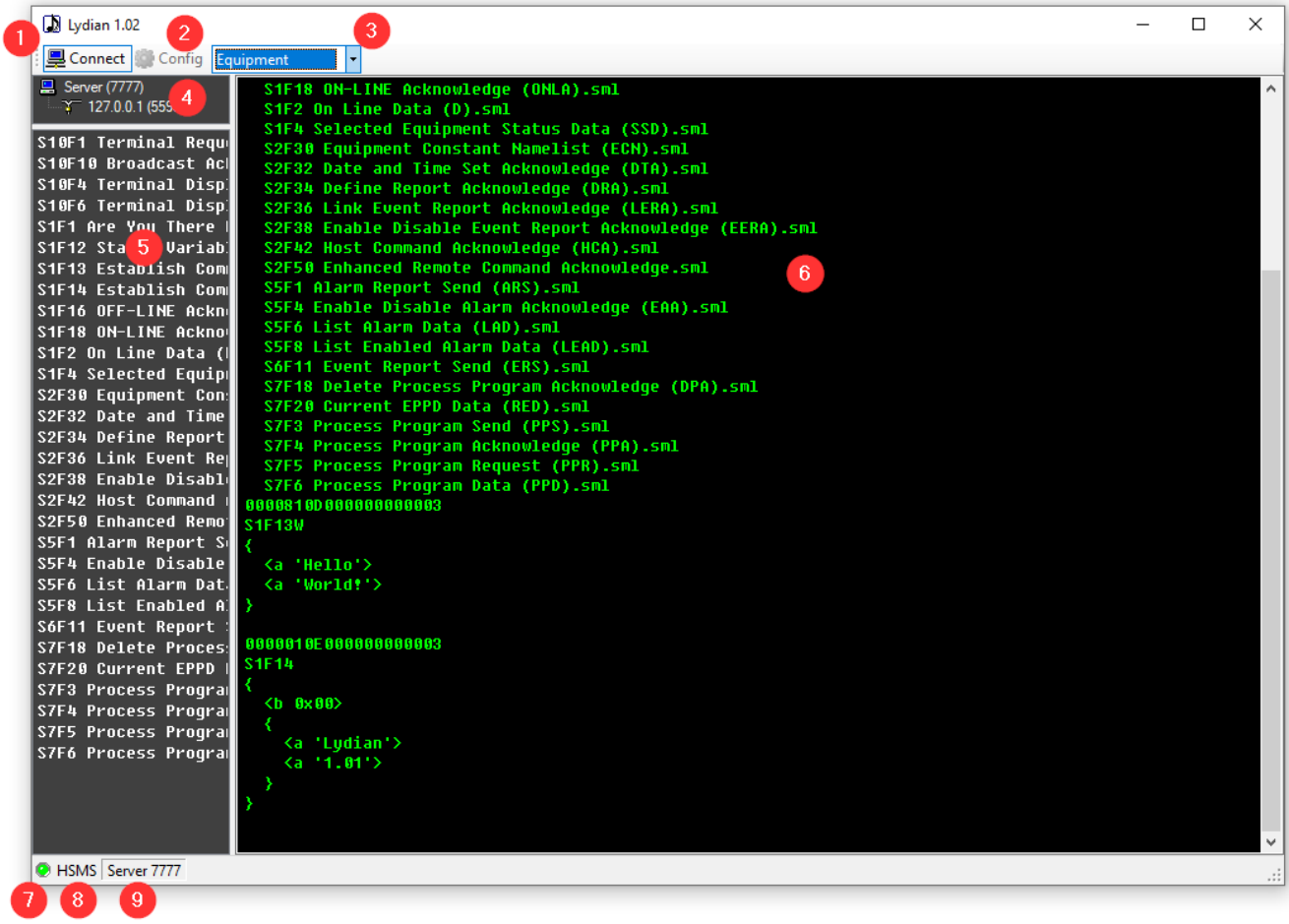


5 「New」ボタンを押してアセンブリのあるフォルダを入力します。



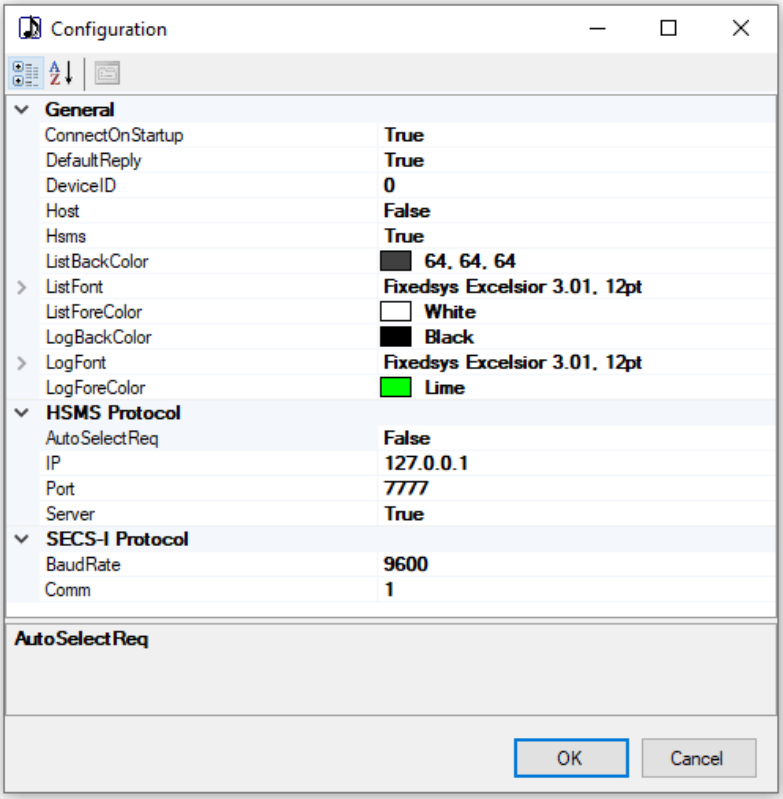
4. ユーザーインターフェース

4.1. メイン画面



番号	説明
1	**接続ボタン**を押すと物理層を有効または無効にします。物理層は HSMS または SECS-I です。有効になるとこのボタンは青色の枠付き（チェック状態）で表示されます。HSMS パッシブエンティティ（サーバ）の場合は、指定された TCP ポートをオープンし、アクティブエンティティ（クライアント）からの接続を待ちます。
2	**設定ボタン**を押すと設定画面を開きます。接続中はこのボタンは無効になります。
3	**SECS-II メッセージセット**の一覧です。
4	**接続状態**が表示されます。
5	**SECS-II メッセージ**が表示されます。ユーザがメッセージをダブルクリックすると、Lydian はメッセージを送信します。
6	**通信ログ**が表示されます。
7	**通信状態インジケータ**は、接続中は緑、切断中は灰色で表示されます。接続ボタンの周囲に表示される青色の枠と同じです。
8	**物理層**は HSMS または SECS-I です。
9	**接続パラメータ**が表示されます。HSMS はサーバ（パッシブエンティティ）またはクライアント（アクティブエンティティ）で、TCP ポート番号が続きます。SECS-I はシリアルポート番号とボーレートが表示されます。

4.2. 設定ダイアログボックス



4.2.1. General

アイテム	説明
ConnectOnStartup	Lydian 起動時に自動的に接続します。
DefaultReply	もしスクリプトでも返信せず、返信メッセージも登録されていない場合、<b 0> を返信します。
DeviceID	SECS-II のデバイス ID。
Host	Lydian がホスト側として動くか、装置側として動くか。
Hsms	物理層が HSMS か SECS-I か。
ListBackColor	SECS-II メッセージリストの背景色。
ListFont	SECS-II メッセージリストのフォント。
ListForeColor	SECS-II メッセージリストの文字色。
LogBackColor	メッセージログの背景色。
LogFont	メッセージログのフォント。
LogForeColor	メッセージログの文字色。

4.2.2. HSMS Protocol

アイテム	説明
IP	IP アドレス。パッシブエンティティの時は 127.0.0.1 に設定します。
Port	TCP ポート番号。
Server	HSMS パッシブエンティティ（サーバ）またはアクティブエンティティ（クライアント）。

4.2.3. SECS-I Protocol

アイテム	説明
BaudRate	ボーレート（通信速度）。典型的な値は 9600 bps です。
Comm	シリアルポート番号。

5. フォルダ構成

5.1. フォルダ

各シミュレーション環境を保存する場合、個別にフォルダを作るのが最も簡単な方法です。他のシステムの設定が影響することもありません。

```
+-- Lydian
     |
     +-- msgs
     |    |
     |    +-- Equipment
     |    |
     |    +-- Host
     |
     +-- plugins
```

フォルダ	説明
Lydian	Lydian.exe や環境設定が保存されているフォルダ。
Msgs	メッセージが保存されているフォルダ。
plugins	プラグインが保存されているフォルダ。

新しいメッセージセットを作成する場合は、msgs/ フォルダ配下にフォルダを作成します。

5.2. ファイル

5.2.1. Lydian フォルダ

下記のファイルが格納または生成されます。

ファイル	説明
debug.log	ログファイル。
debug001.log	ログのバックアップファイル。
Lydian.config	Lydian の設定。
Lydian.exe	Lydian シミュレータ実行ファイル。

5.2.2. Msgs フォルダ

メッセージファイルはこのフォルダに保存します。ファイルの拡張子は「**.sml**」です。

5.2.3. Plugins フォルダ

プラグインファイルはこのフォルダに保存します。ファイルの拡張子は「**.dll**」です。

6. プラグイン

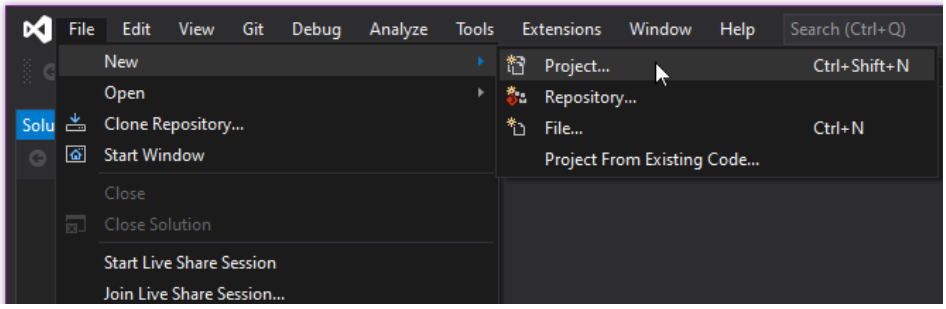
スクリプトはプラグインとして作ります。Lydian は複数のプラグインをインポートすることが可能です。プラグインは C#、Visual Basic またはその他の.NET 対応言語で作成することができます。

6.1. C#チュートリアル

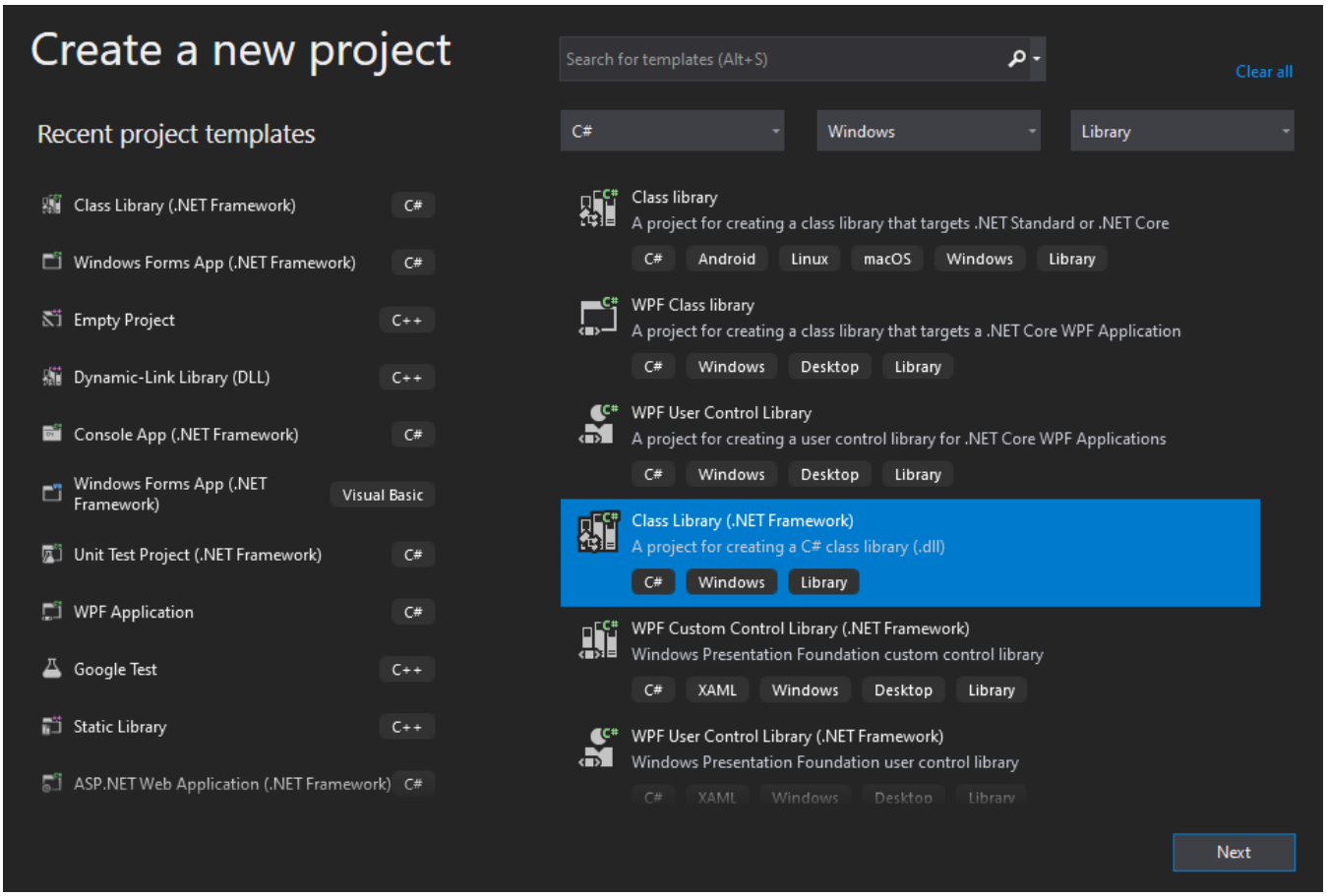
実際に Visual Studio 2019 と Windows 10 を使ってプラグインを作ってみましょう。この手順は Visual Studio 2022 や Windows 11 でも同じです。過去のバージョンの Visual Studio と Windows の組み合わせでも同じです。

6.1.1. プラグインプロジェクトの作成

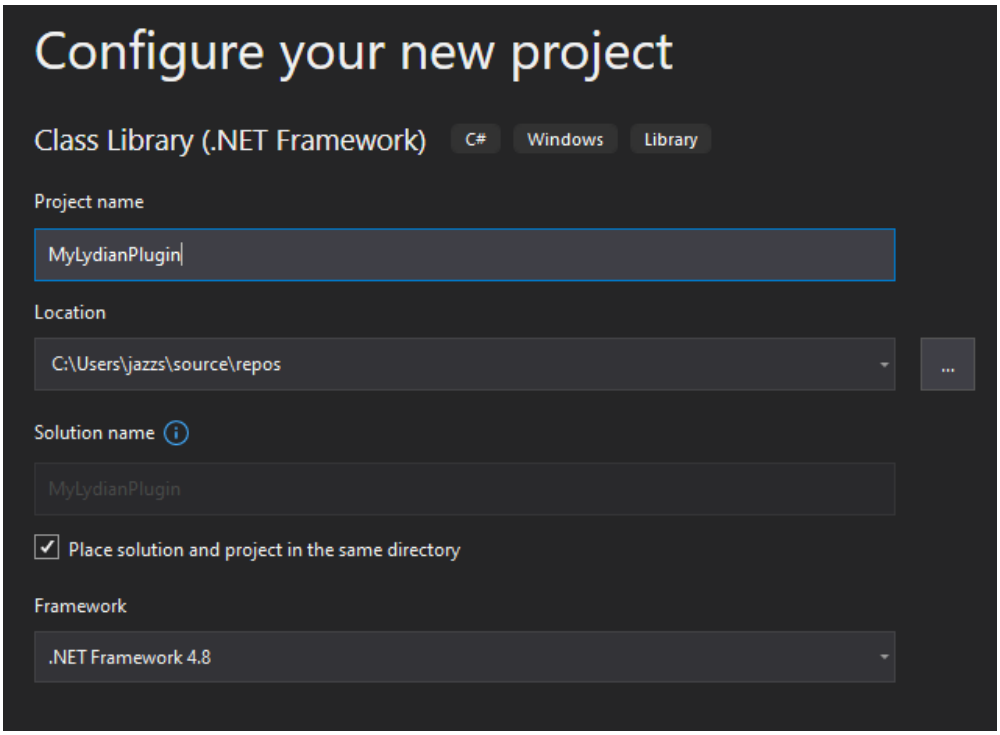
6 [**File**] – [**New**] – [**Project...**]メニューから新規プロジェクトを作成します。



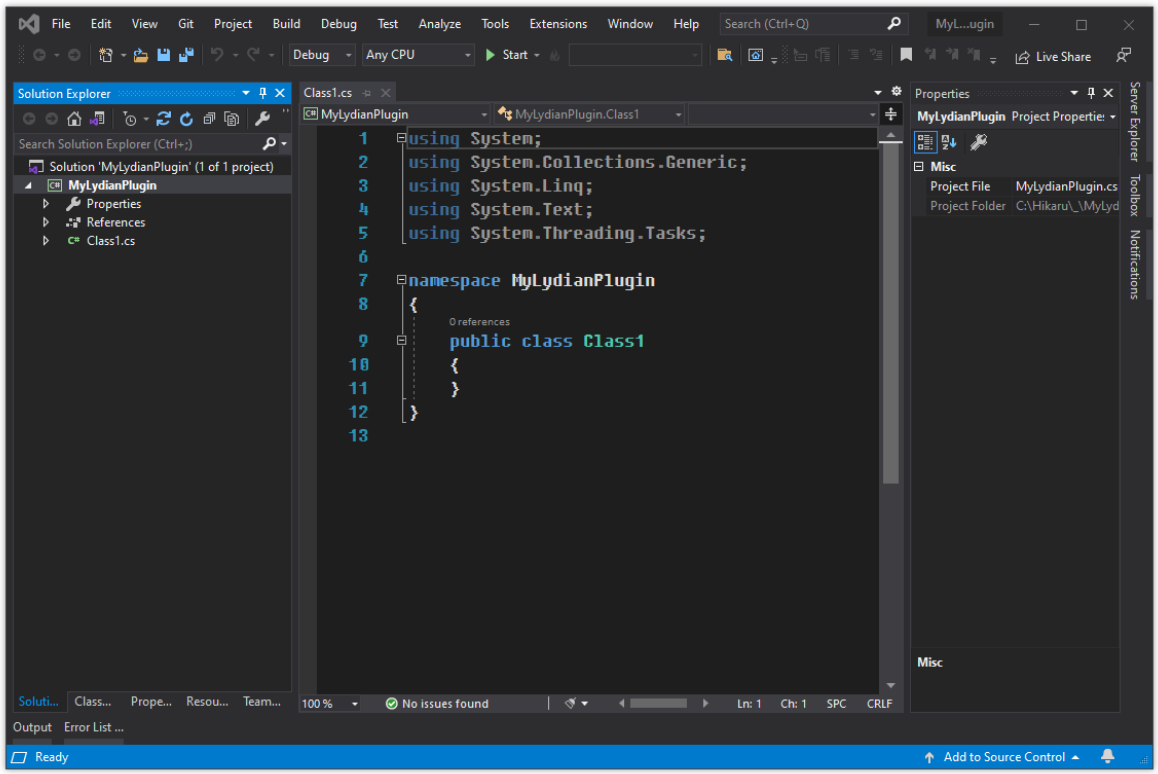
7 ドロップダウンリストから C# – Windows – Library を選択します。「**Class Library (.NET Framework)**」プロジェクトを選択します。非常によく似た名前のプロジェクトタイプがあるので要注意です。



8 プロジェクト名を入力し、プロジェクトを作るフォルダ、.NET Framework のバージョンを選択します。個人的には「Place solution and project in the same directory」（プロジェクトファイルとソリューションを同じフォルダに置く）が好みです。これで全て1つのフォルダに入るからです。そして Create ボタンを押します。



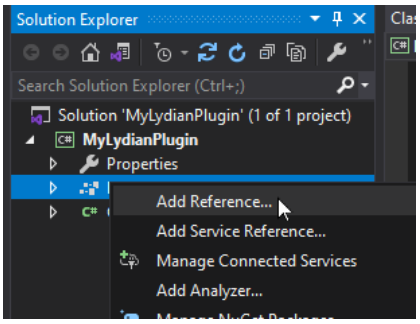
9 プラグインプロジェクトが生成されました。



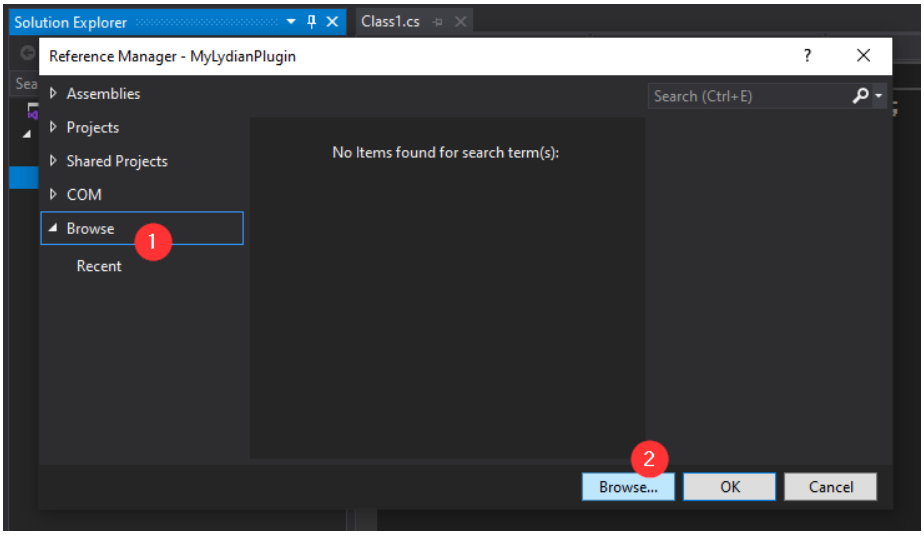
6.1.2. Dorian アセンブリの追加

Lydian は Dorian を使用しています。Dorian.NET.dll の中には Lydian のプラグインのベースとなるクラス情報も入っています。

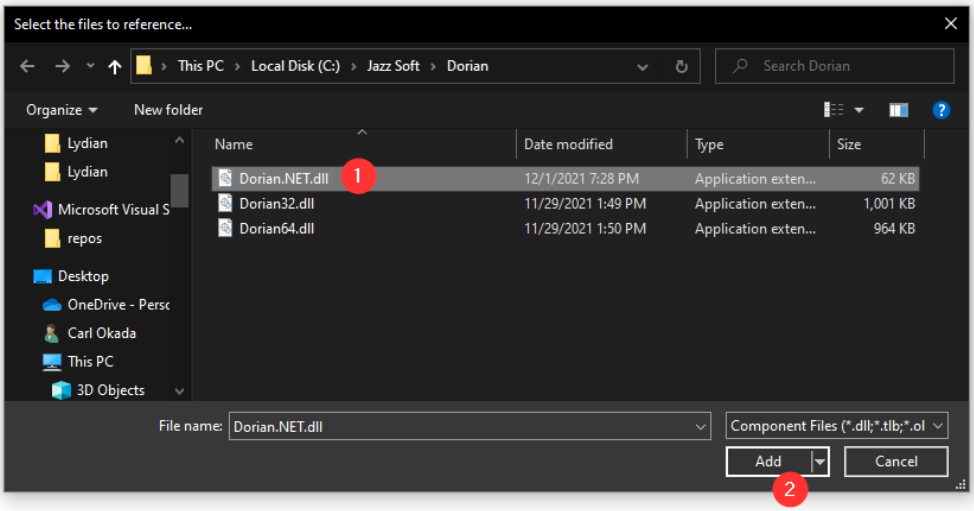
1 ソリューションエクスプローラから **References** を右クリックします。そして **Add Reference...**を選択します。



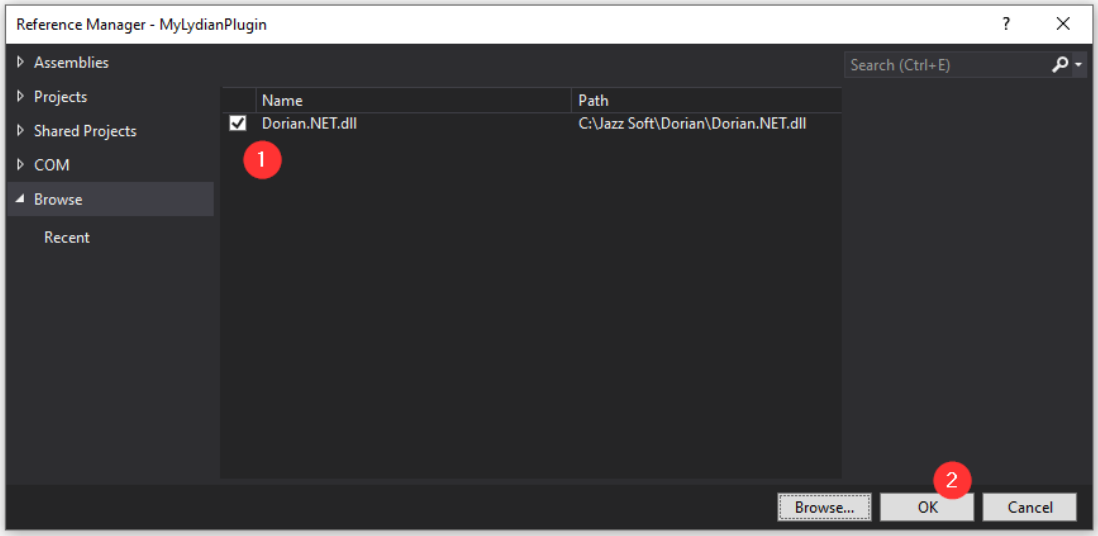
2 左ペインから **Browse** を選択し、**Browse...**ボタンを押します。



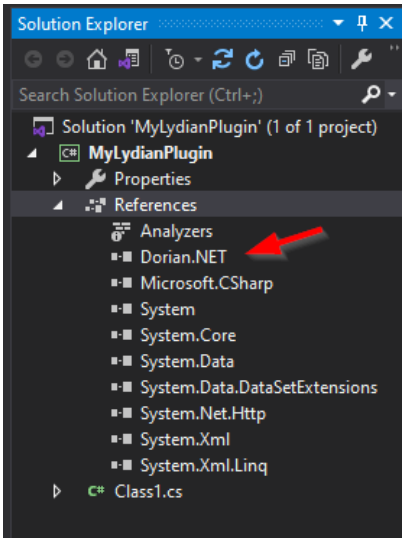
3 **C:¥Jazz Soft¥Dorian¥**フォルダに行って **Dorian.NET.dll** を選択します。そして **Add** ボタンを押します。



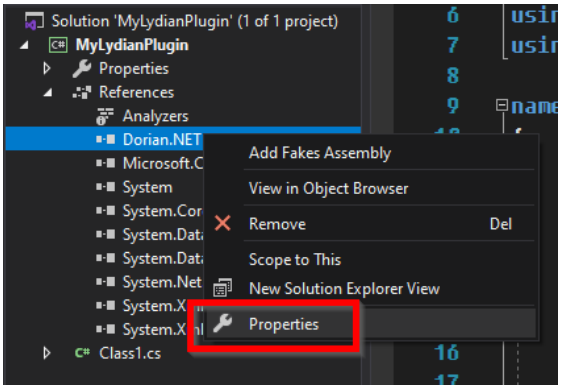
4 念のため **Dorian.NET.dll** がチェックされていることを確認します。そして OK ボタンを押します。



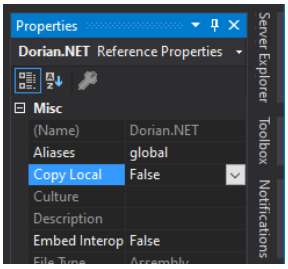
5 References に Dorian.NET が追加されました。



6 右クリックして **Properties** を選択します。

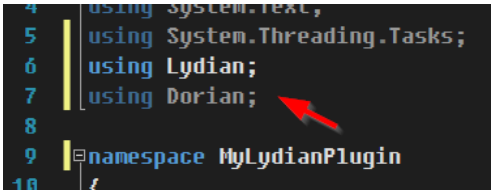


7 **Copy Local** を **False** に変更します。コピーする必要がないからです。

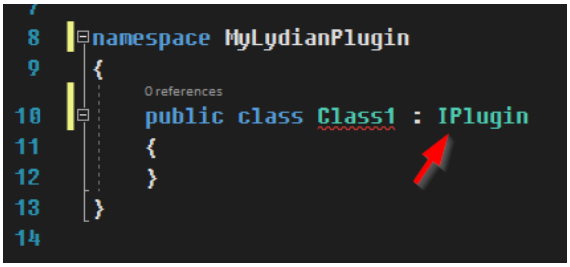


6.1.3. Lydian.Plugin から継承させる

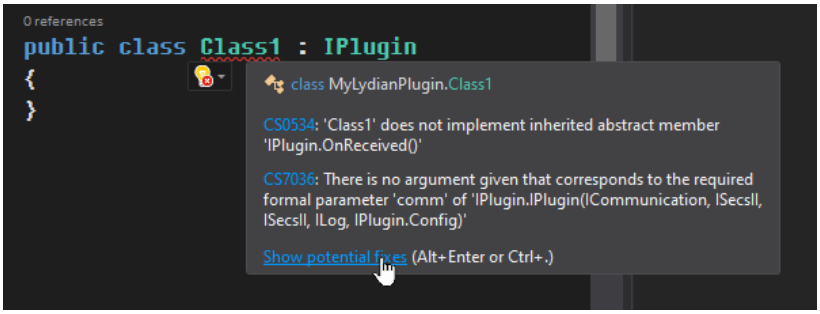
1 手間を省くため「**using Lydian;**」と「**using Dorian;**」を追加します。これは必須ではありません。



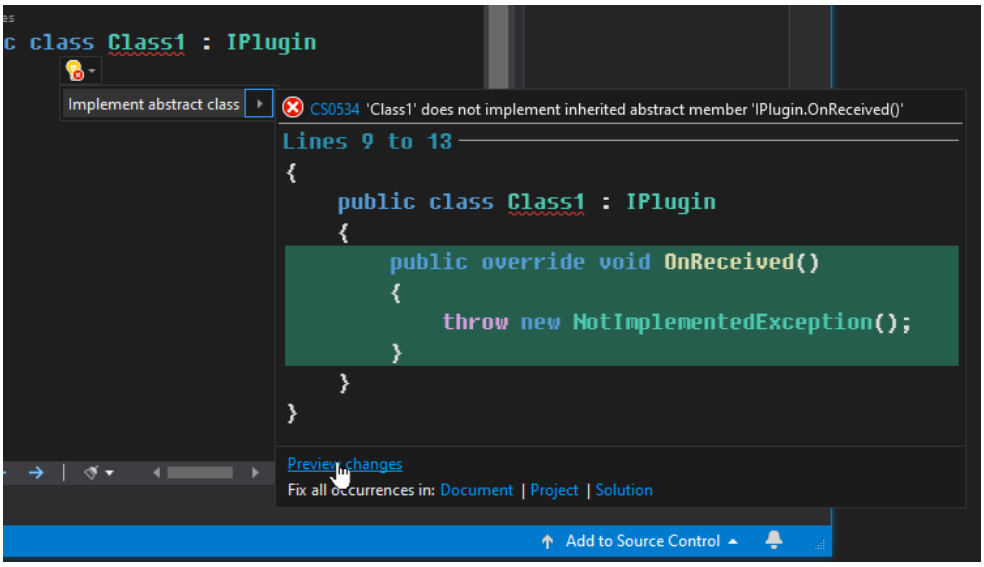
2 Class1 クラスを **Lydian.IPlugin** クラスから継承させます。



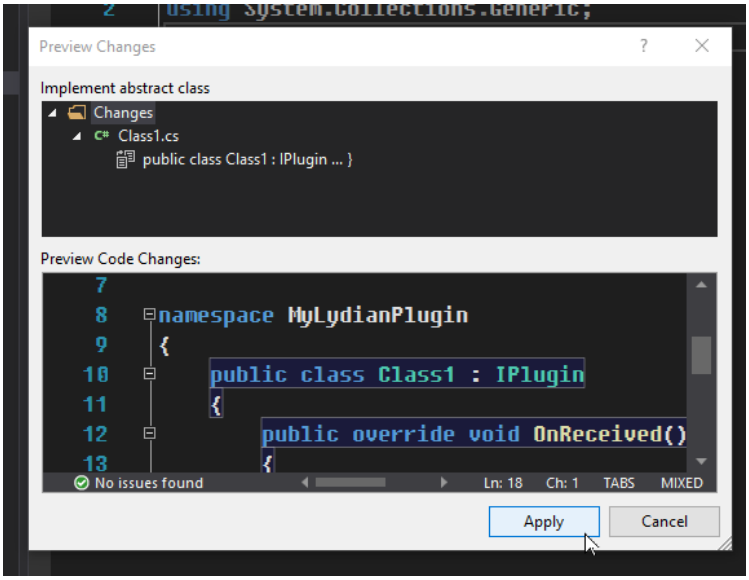
3 Lydian.IPlugin は抽象クラスです。必要な関数を実装する必要があります。それが理由で赤線が表示されています。クラス名のところにマウスを持っていき、ポップアップメッセージから **Show potential fixes** をクリックします。



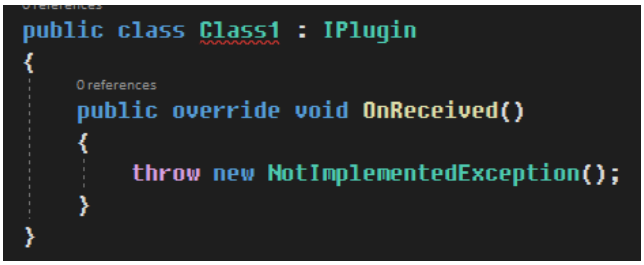
4 **Preview changes** をクリックします。



5 **Apply** ボタンをクリックします。.



6 **OnReceived**()関数が実装されましたが、クラス名の下にはまだ赤線が表示されています。Lydian.IPlugin は依存性の注入のため、指定された形式のコンストラクタが必要だからです。



7 下記のコンストラクタのコードをクラスに追加します。これでエラーは消えました。

```
public Class1(LydianObj lydian)
```

```
    : base(lydian)
{
}
```

```
0 references | 0 changes | 0 authors, 0 changes
public Class1(LydianObj lydian)
    : base(lydian)
{
}
```

6.1.4. 情報の追加

1 **log** オブジェクトを直接操作することでメッセージログに書き出すことができます。基底クラス（IPlugin 抽象クラス）のメンバーにコピーされるため、いつでも使えるようになります。

```
0 references | 0 changes | 0 authors, 0 changes
public Class1(LydianObj lydian)
    : base(lydian)
{
    lydian.log.Write("My Lydian plugin has been loaded!");
}
```

6.1.5. メッセージに返信する

Lydian がメッセージを受信すると OnReceived()が呼ばれます。S1F13 に返信してみましょう。

1 受信メッセージは Lydian.IPlugin クラスの **msgi** プロパティにセットされます。Lydian.IPlugin クラスから継承しているため、**msgi** プロパティに直接アクセスできます。

```
if (lydian.msgi.Stream == 1 && lydian.msgi.Function == 13)
{
    // S1F13
}
```

2 送信メッセージは **msgo** プロパティにセットする必要があります。メッセージを送信するには Lydian.IPlugin クラスの **Send**()メソッドを使います。

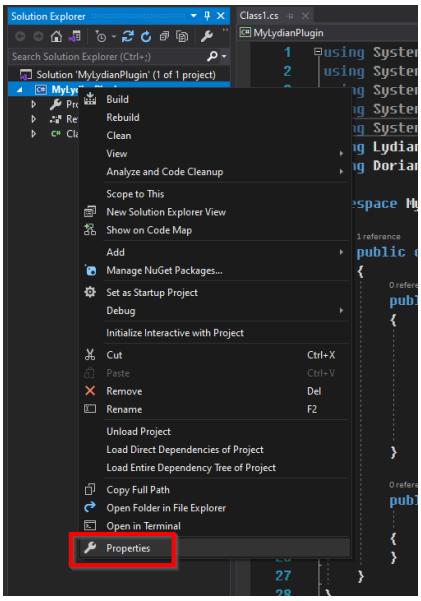
```
    lydian.msgo.Sml = "{<b 0>{}}";
    Send(lydian.msgi.Msg);
```

3 OnReceived()は以下のようになります。

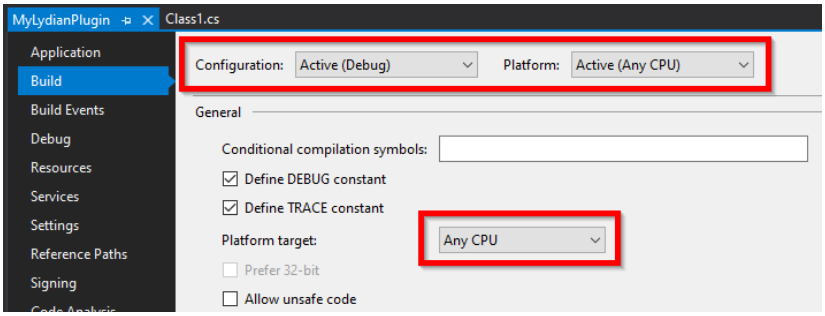
```
0 references | Carl Okada, 6 hours ago | 1 author, 1 change
public override void OnReceived()
{
    if (lydian.msgi.Stream == 1 && lydian.msgi.Function == 13)
    {
        // S1F13
        lydian.msgo.Sml = "{<b 0>{}}";
        Send(lydian.msgi.Msg);
    }
}
```

6.1.6. デバッグ環境のセットアップ

1 前述のように、プラグイン DLL は「plugin」フォルダに格納する必要があります。Visual Studio で DLL の出力フォルダを変更してみましょう。output folder on Visual Studio.　プロジェクトを右クリックし、**Properties** を選択します。

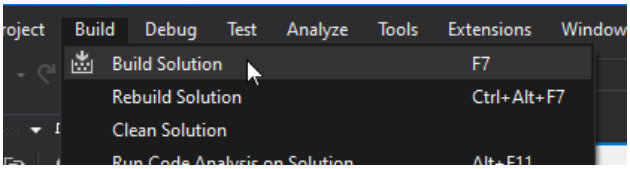


2 左ペインから **Build** を選びます。ビルド設定で、Configuration が「**Debug**」、Platform が「**Any CPU**」であることを再確認した上で、Platform target が「**Any CPU**.」であることを確認します。

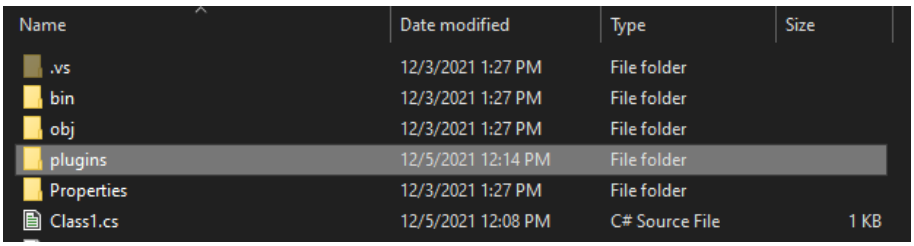


3 同じ画面で Output path を「**plugins¥**」に変更します。

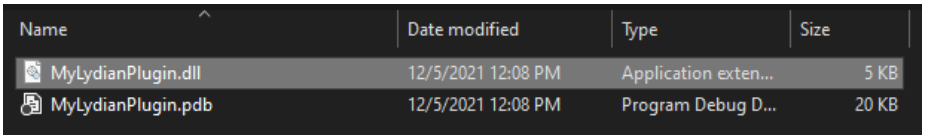
4 プロジェクトをビルドします。メニューから、[**Build**] – [**Build Solution**]を選択します。



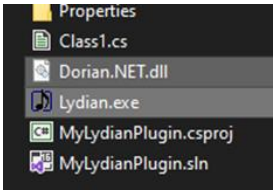
5 **plugins** フォルダが作られました。



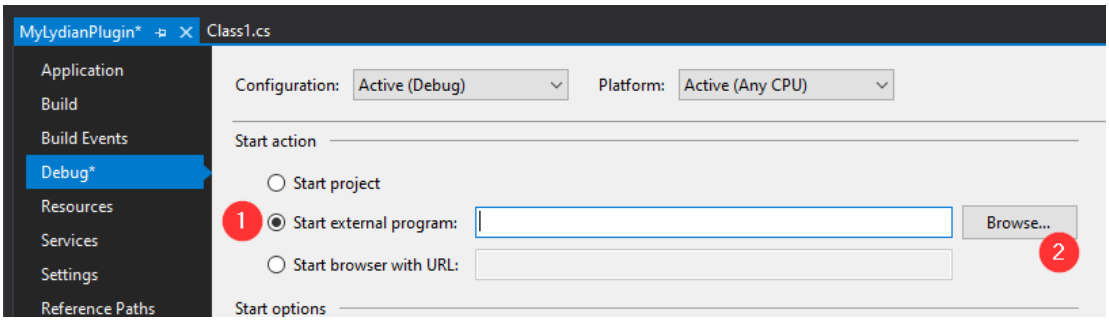
6 **plugins** フォルダの中には、プラグイン DLL が生成されています。.



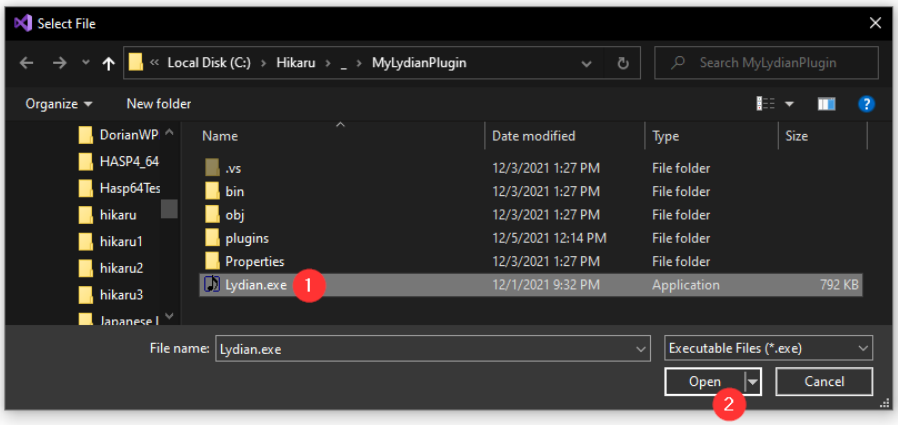
7 **C:¥Jazz Soft¥Lydian¥**フォルダに行き、**Lydian.exe** と **Dorian.NET.dll** をプロジェクトフォルダにコピーします。



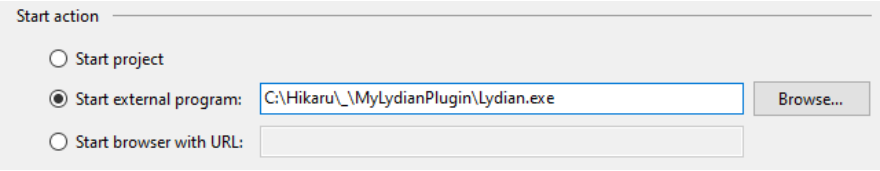
8 プロジェクト画面に戻り、左ペインから **Debug** を選択します。**Start external program** を選択して **Browse...**ボタンを押します。



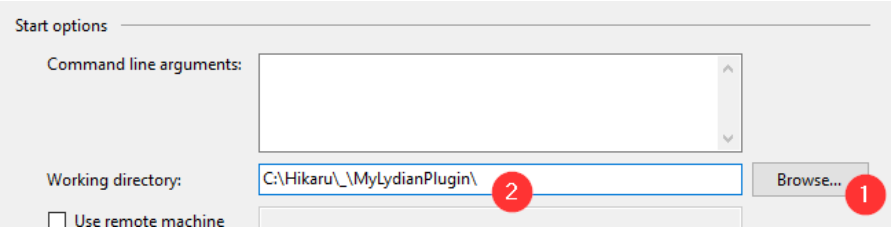
9 **Lydian.exe** を選び、**Open** ボタンを押します。



10 Now Lydian.exe is selected for debugging.

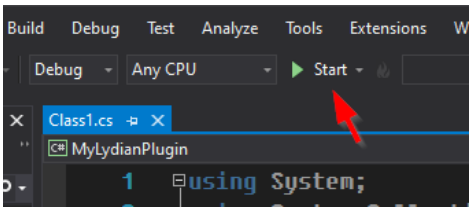


11 On the same page, do the same thing for **Working directory**. This has to be the same folder as Lydian.exe.



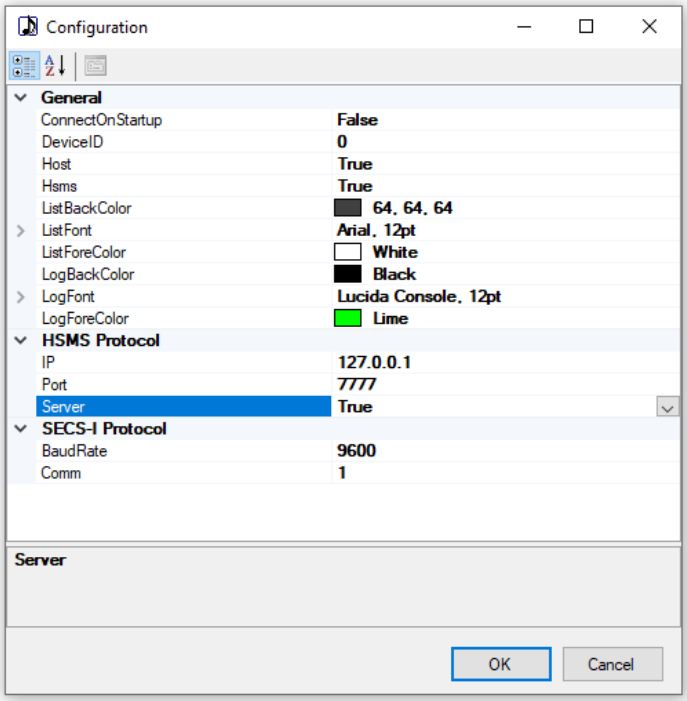
6.1.7. Lydian でプラグインをデバッグする

1 **Debug** と **Any CPU** が選択されていることを確認し、Visual Studio で **Start** ボタンを押します。

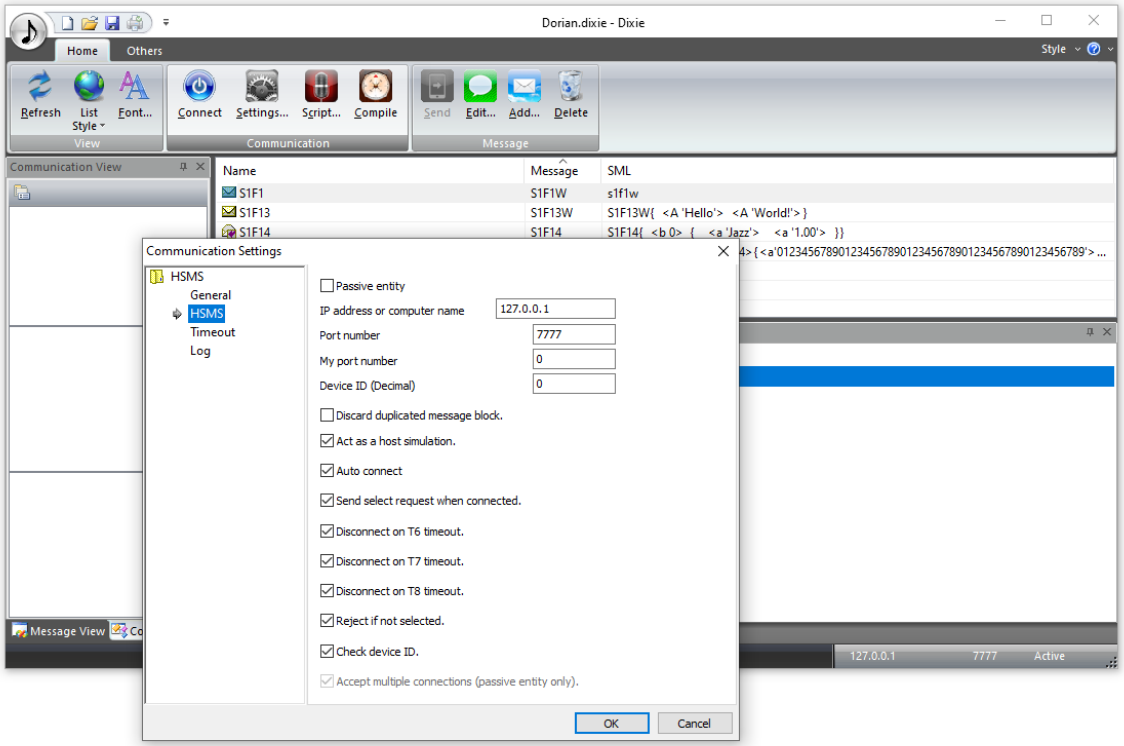


2 Lydian が起動しますが、通信パラメータはまだ設定されていません。**Config** ボタンを押します。

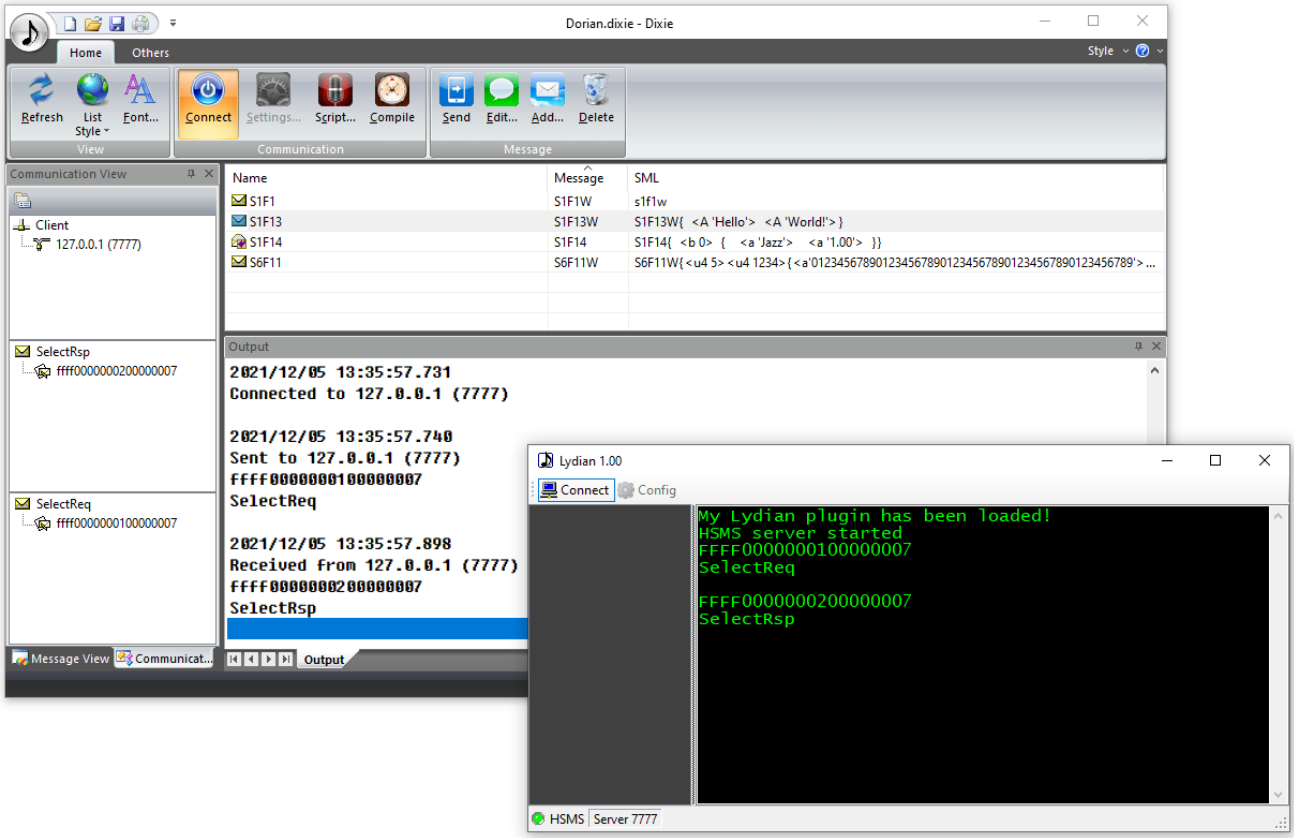
3 パラメータを設定します。このチュートリアルでは、HSMS サーバで TCP ポート番号を 7777 に設定します。



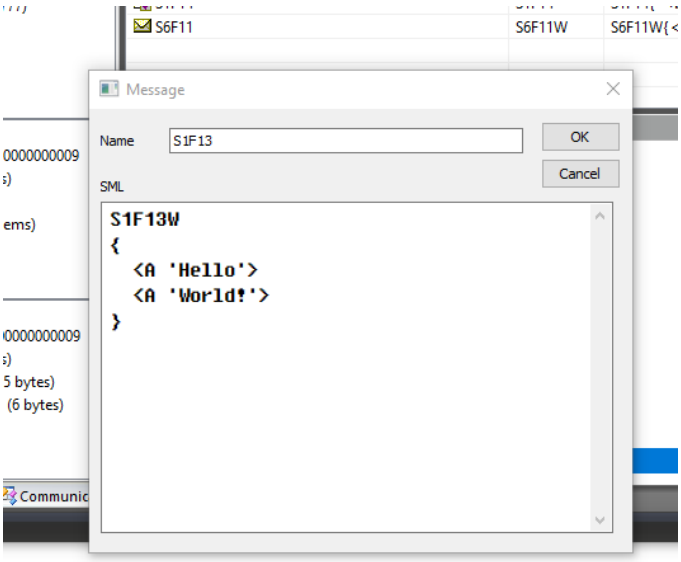
4 Dixie シミュレータ（もしくはお好みの HSMS シミュレータアプリケーション）を起動し、HSMS クライアントに設定します。



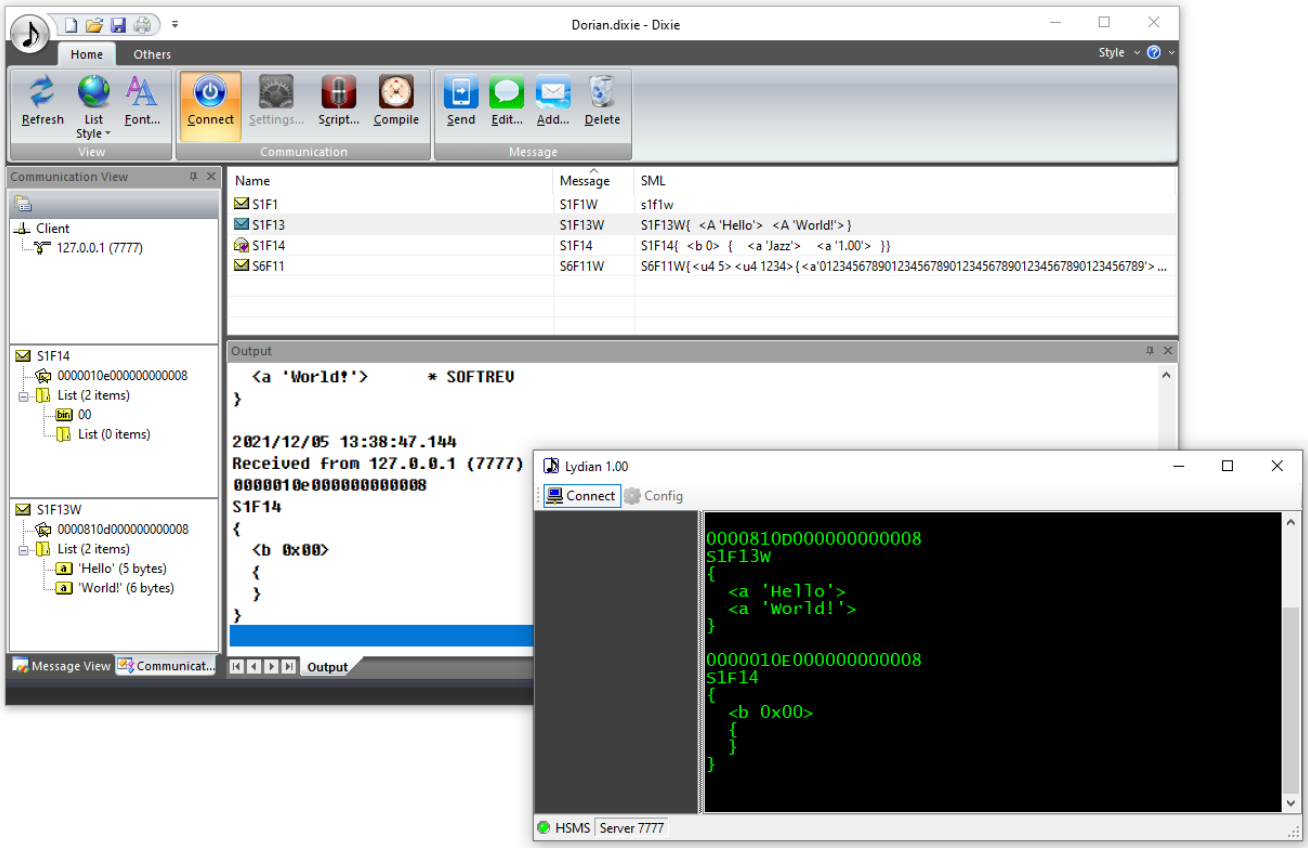
5 Connect ボタンを押します。接続が開始します。



6 S1F13 は下記のようになります。



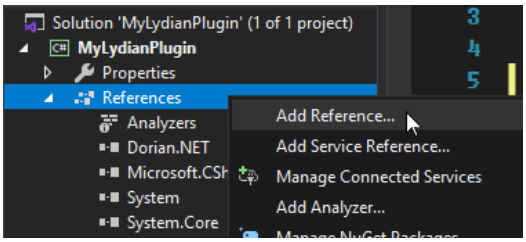
7 S1F13 を送信すると、プラグインが S1F14 を返信します。ソースコードでブレイクポイントをセットすることもできます。



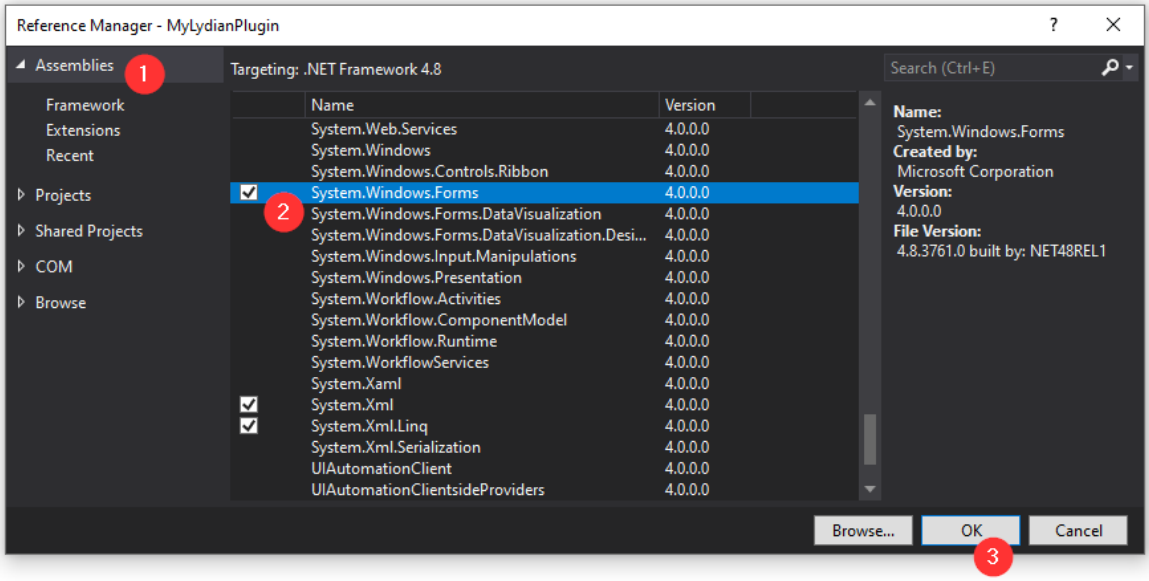
6.2. タイマーを使う

6.2.1. 一定周期で S1F1 を送信する

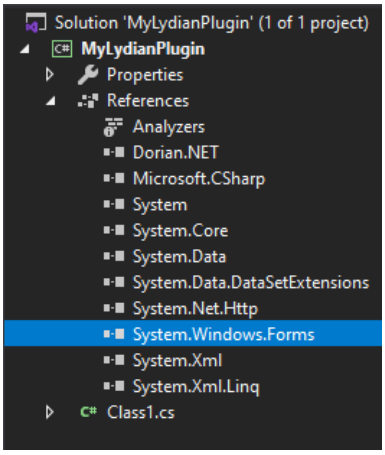
1 .NET ライブラリの中にはタイマーがいくつかありますが、**System.Windows.Forms.Timer** だけがマルチスレッドを気にしないで気軽に使用することができます。**References** を右クリックし、**Add Reference...**を選択します。



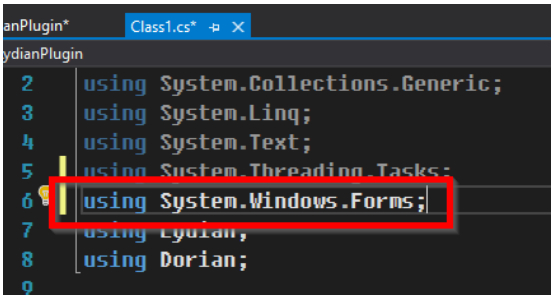
2 左ペインから **Assemblies** を選択し、**System.Windows.Forms** にチェックを入れます。



3 System.Windows.Forms が追加されました。



4 「**using System.Windows.Forms;**」をソースコードに追加します。これは必ずしも必要ではありません。



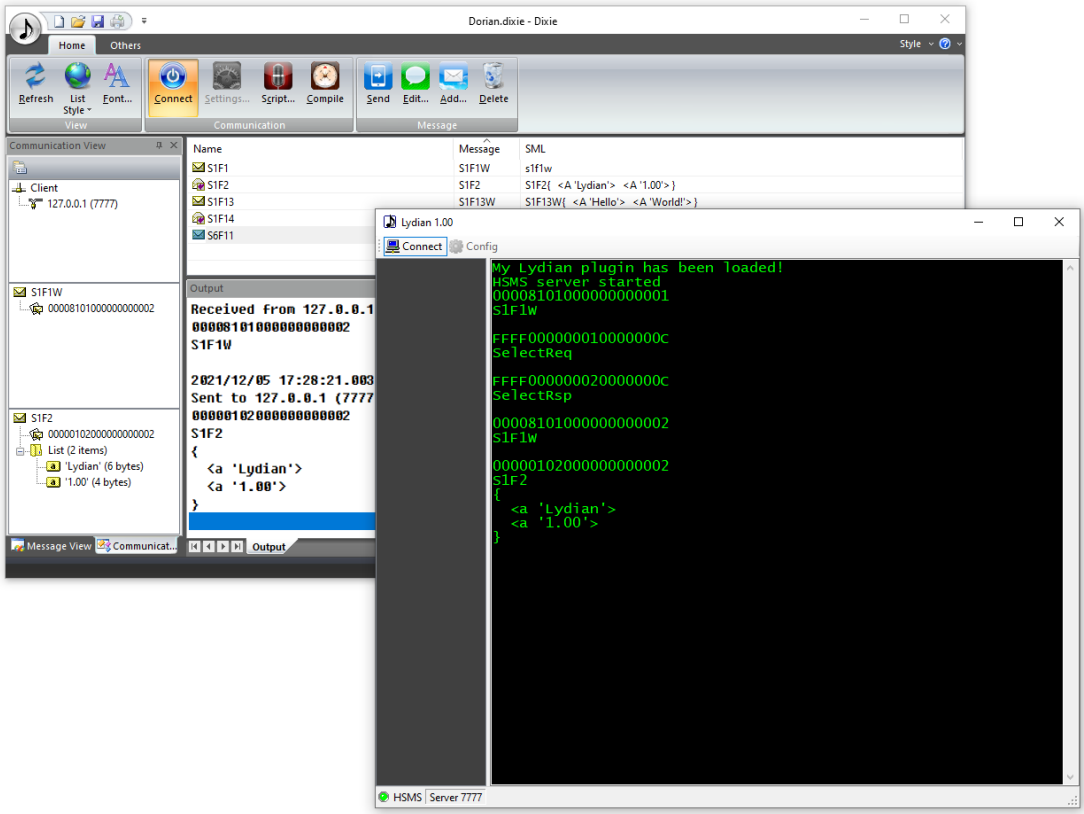
5 クラス中にタイマーを作成します。タイマー周期は 10 秒です。

```
protected Timer timer = new Timer { Enabled = true, Interval = 10 * 1000 };
```

6 比較的新しい技術である「ラムダ式」を使ってみましょう。タイマーでしか使用することのない新たに関数を作成する必要がなくなります。

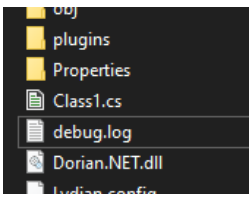
```
timer.Tick += (sender, e) =>
{
    if (!lydian.comm.Connect)
        return;

    lydian.msgo.Sml = "S1F1W";
    Send();
};
```

7 プラグインは 10 秒おきに S1F1 を送信します。

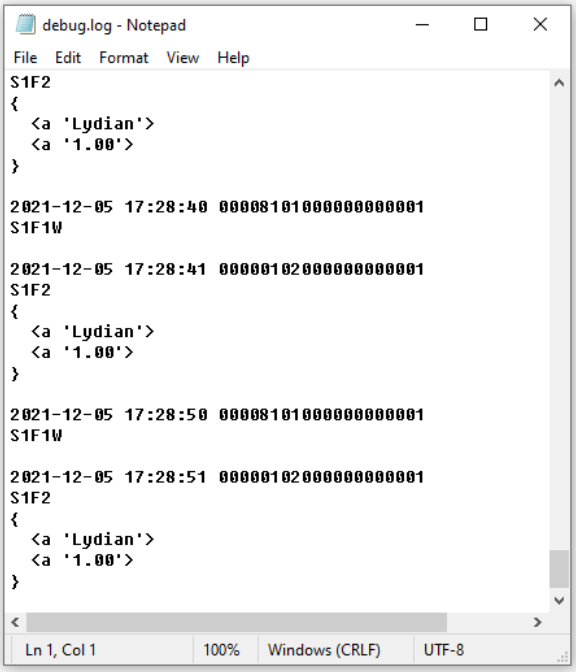


6.2.2. ログファイル

1 ログファイルは **debug.log** という名前で作られます。



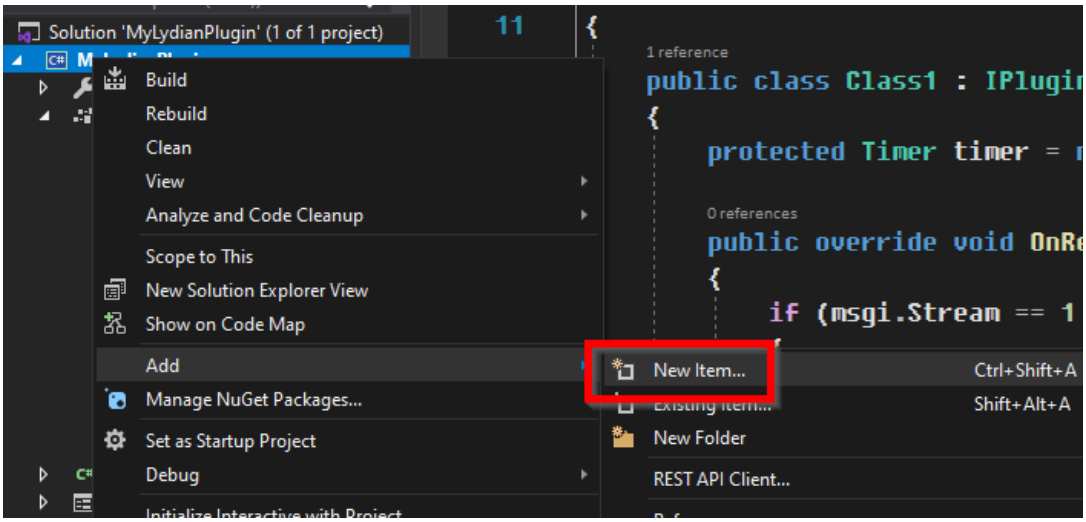
2 日付と時刻が追加されています。



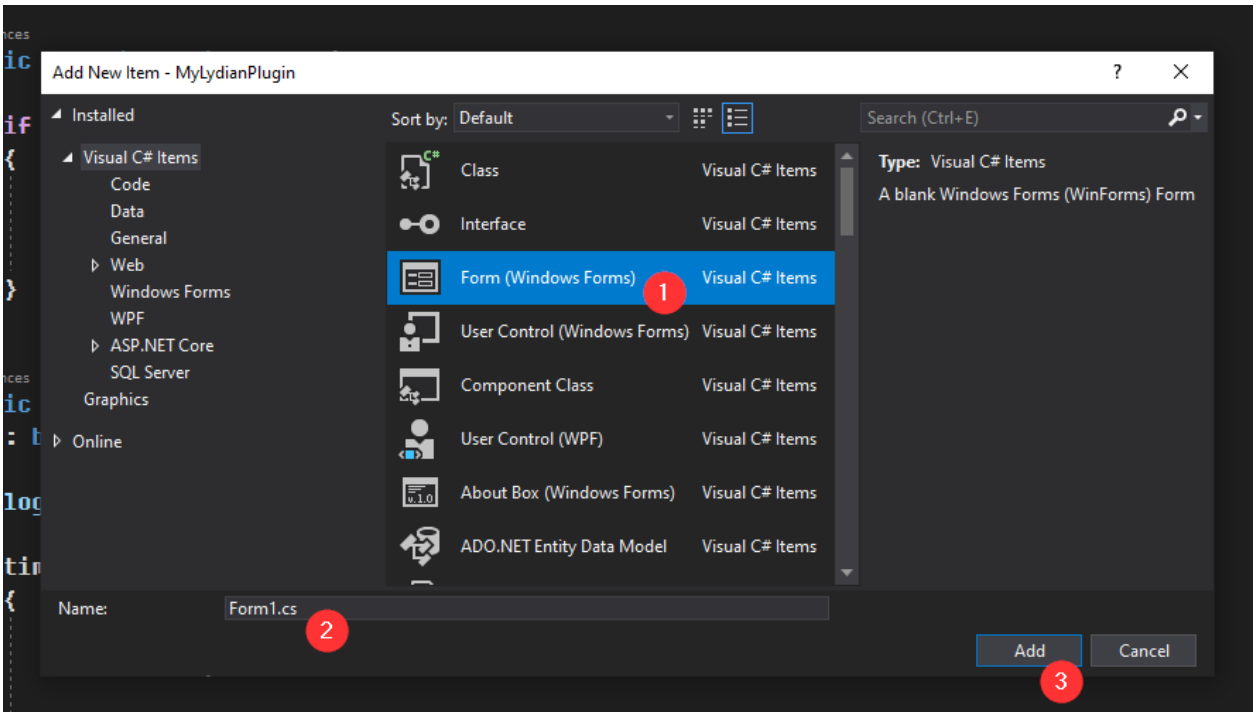
6.3. ユーザインターフェースを追加する

6.3.1. ダイアログボックスの追加

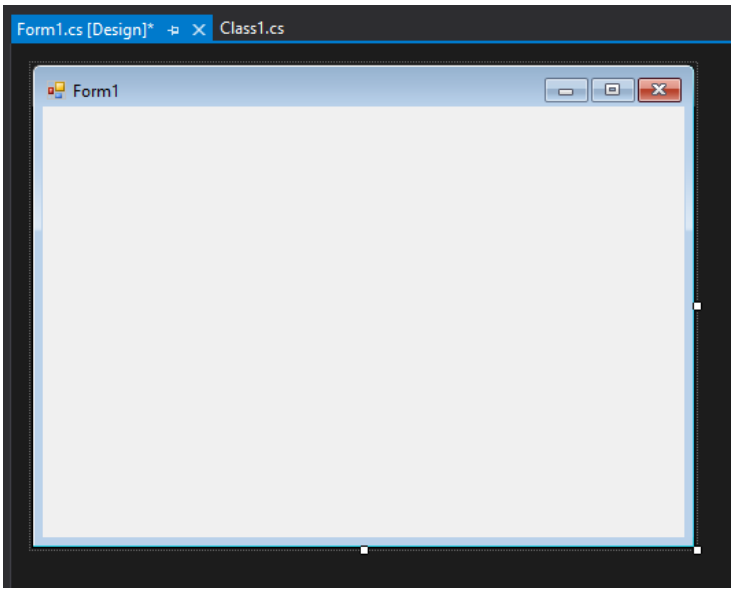
1 プロジェクトを右クリックし、**Add** – **New Item...**を選択する。



2 **Form (Windows Forms)**を選択し、名前をつけて **Add** ボタンを押します。



3 フォームがプロジェクトに追加されました。



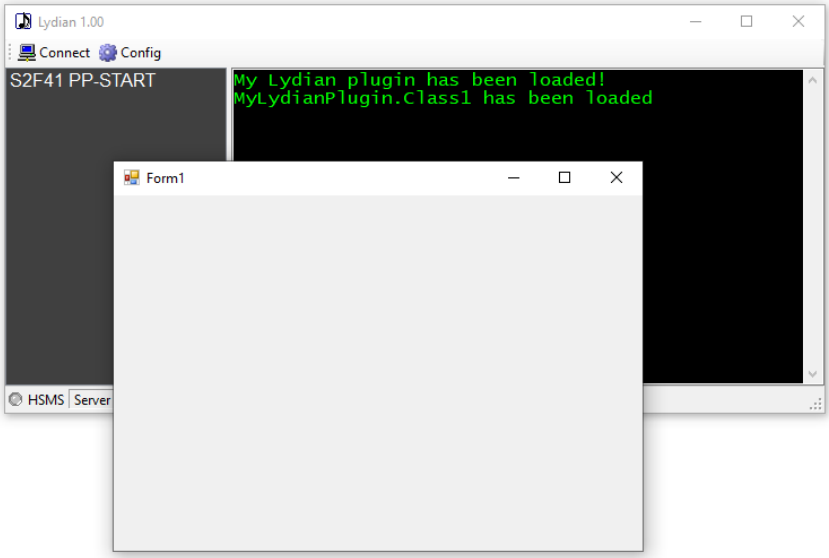
4 フォームのオブジェクトを生成します。

```
1 reference
public class Class1 : IPlugin
{
    protected Timer timer = new Timer { Enab
    protected Form1 form = new Form1();
```

5 フォームの Show()メソッドを呼びます。

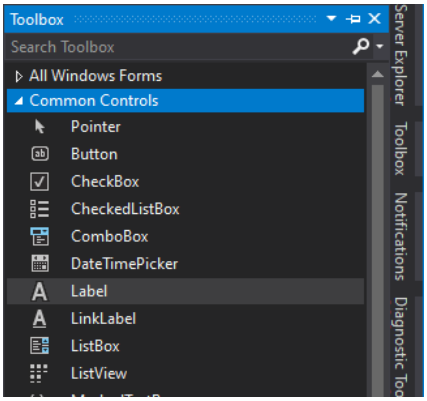
```
0 references | 0 changes | 0 authors, 0 changes
public Class1(LydianObj lydian)
    : base(lydian)
{
    lydian.log.Write("My Lydian plugin has been loaded!");

    timer.Tick += (sender, e) =>
    {
        if (!lydian.comm.Connect)
            return;

        lydian.msgo.Sml = "S1F1W";
        Send();
    };

    form.Show();
}
```

6 モードレスダイアログボックスが表示されます。

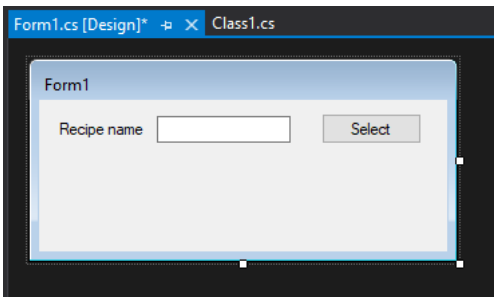


6.3.2. レシピを選択する

1 ツールボックスを開きます。



2 ラベル、テキストボックス、ボタンを配置します。コントロールボックスは非表示にします。



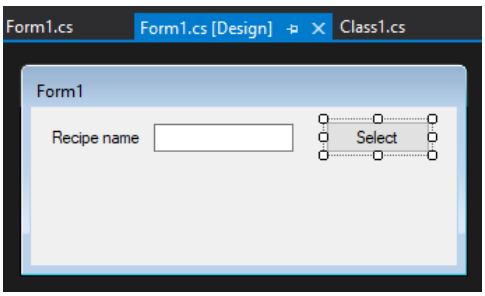
3 Form1 で Lydian.IPlugin オブジェクトの参照を受け取れる準備をします。

```
4 references
public partial class Form1 : Form
{
    public Lydian.IPlugin plugin;

    1 reference
    public Form1()
    {
```

4 参照を Class1 から Form1 に渡します。注意点として、オブジェクトを複製するのではなく、実体は Lydian 内にあります。

```
0 references | 0 changes | 0 authors, 0 changes
public Class1(LydianObj lydian)
    : base(lydian)
{
    lydian.log.Write("My Lydian plugin has been loaded!");

    timer.Tick += (sender, e) =>
    {
        if (!lydian.comm.Connect)
            return;

        lydian.msgo.Sml = "S1F1W";
        Send();
    };

    form.plugin = this;
    form.Show(lydian.handle);
}
```

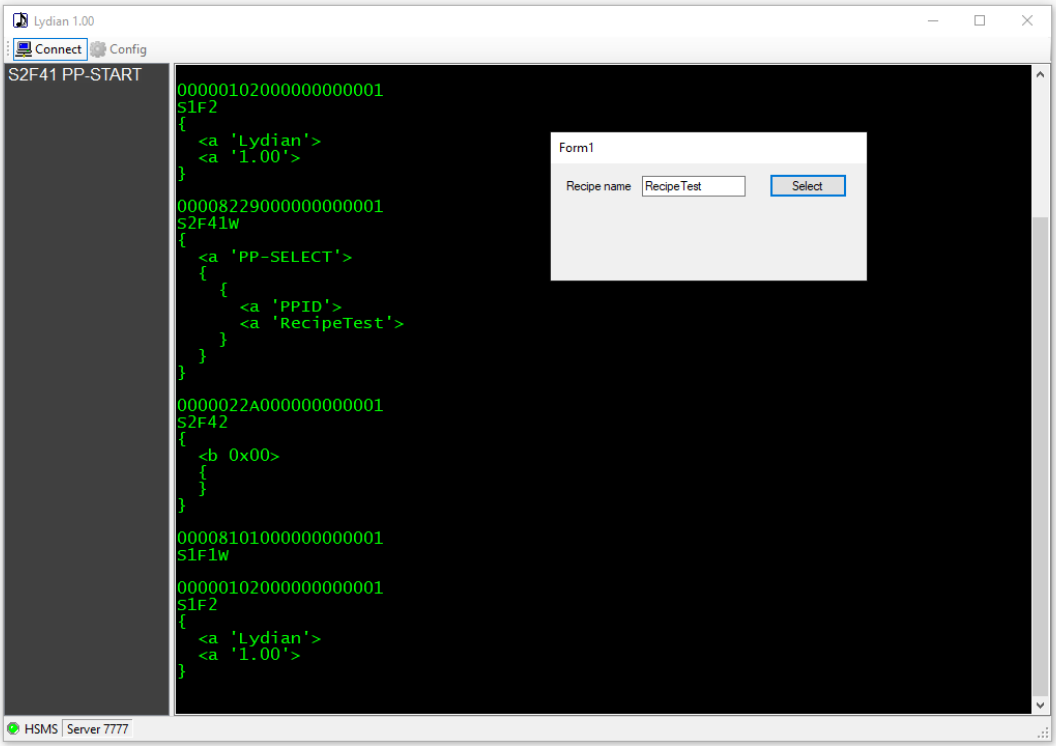
5 Select ボタンをダブルクリックし、イベントハンドラ関数を作成します。



6 PP-SELECT コマンドを送信する SML を書きます。

```
1 reference
private void btnSelect_Click(object sender, EventArgs e)
{
    plugin.msgo.Sml =
        "S2F41W" +
        "{" +
        "   <A 'PP-SELECT'>" +
        "   {" +
        "       {" +
        "           <A 'PPID'>" +
        "           <A '" + recipe.Text + "'>" +
        "       }" +
        "   }" +
        "}";
    plugin.Send();
}
```

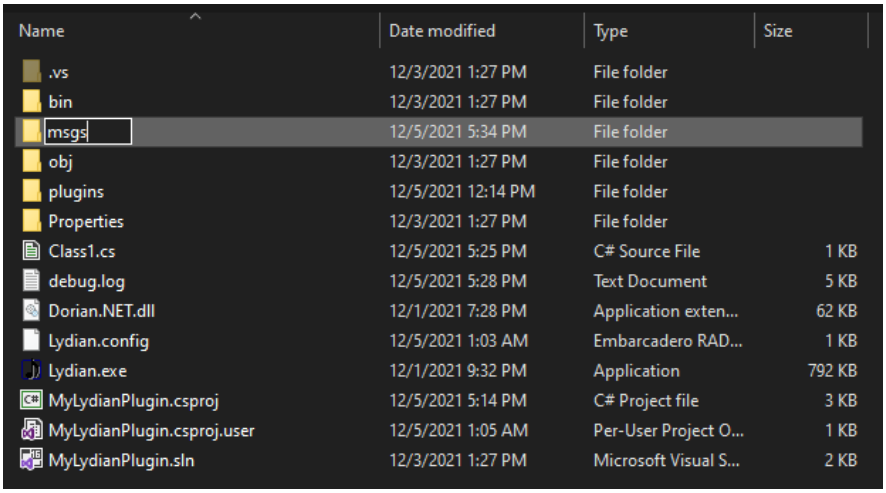
7 プラグインをビルドし、Lydian を実行します。Select ボタンを押すと、PP-SELECT コマンドが送信されます。



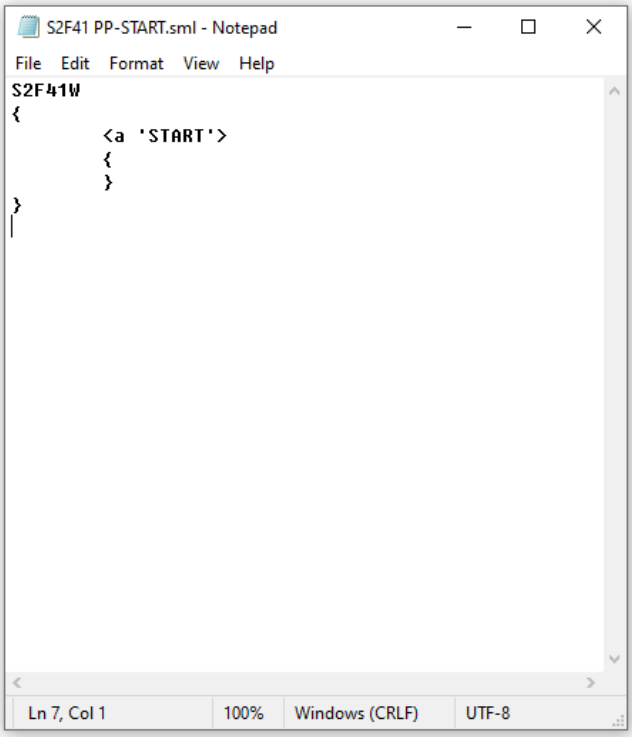
7. メッセージ

7.1. メッセージを追加する

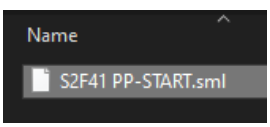
1 **msgs** サブフォルダを作成します。



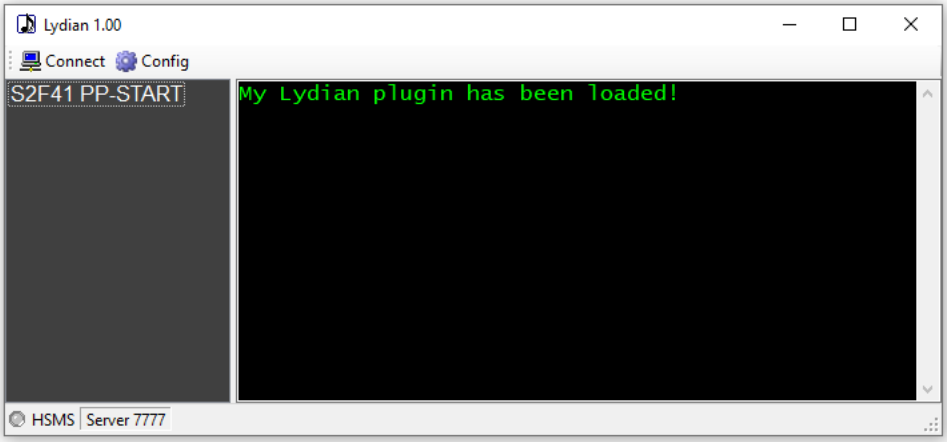
2 メモ帳、Visual Studio、VS Code などのテキストエディタを使って SML でメッセージを記述します。



3 ファイルに保存します。拡張子は「**.sml**」にします。

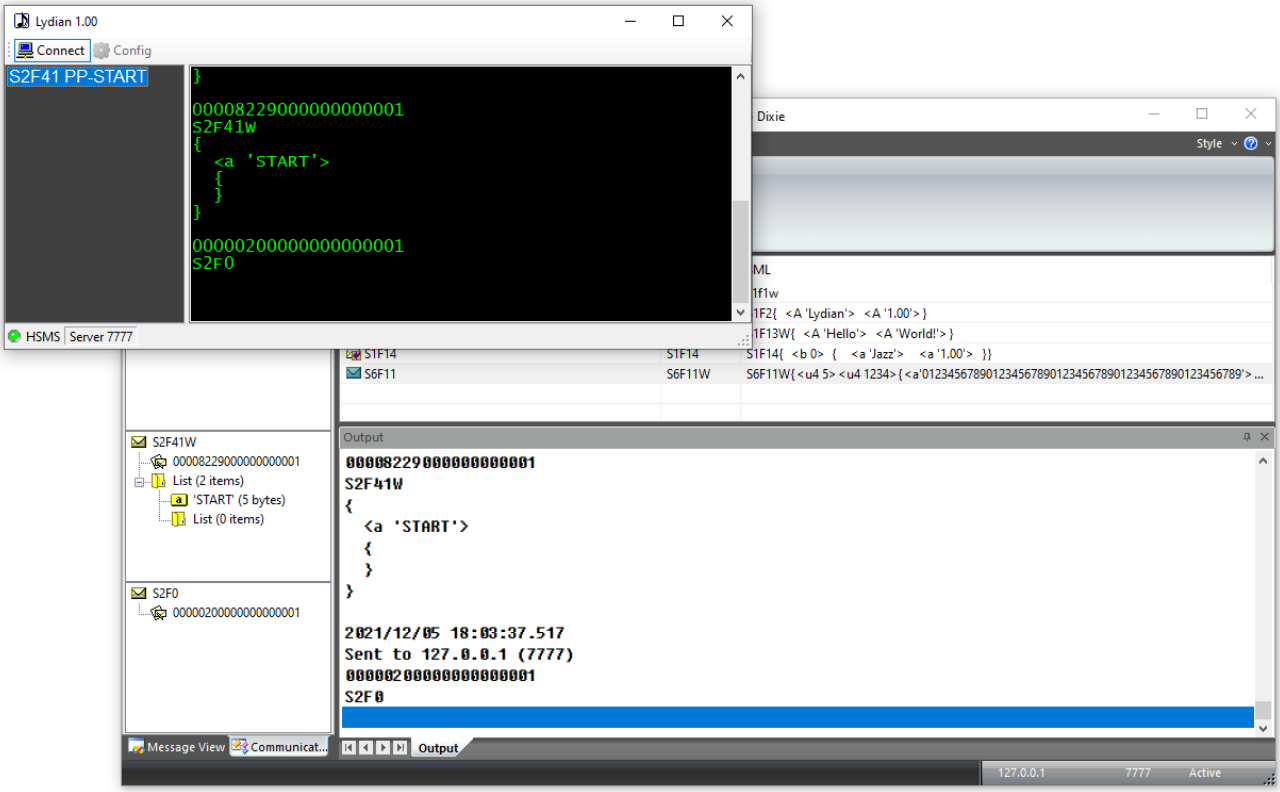


4 Lydian を再起動すると、追加したメッセージが表示されます。



7.2. メッセージを送信する

1 メッセージをダブルクリックすると送信されます。



7.3. SML リファレンス

Sml プロパティにセットする文字列の構文は以下のようになっています。

7.3.1. 一般的な注意

ホワイトスペース（スペース、タブ、改行、復改コード）は区切り文字としての意味しかありません。このため適度にタブや改行コードを挿入することで見易くすることができます。ただしコメント中および文字列中は文字として扱われます。

アスタリスク(*)から行末まではコメントとなります。ただし文字列中のアスタリスクは除きます。

整数は 0～9 までの文字とマイナス(-)から構成されます。16 進数で記述したい場合は' 0x' を先頭に付加します。この場合は a～f と A～F までの文字も使用できます。小数は' 0.9' を' .9' というように' 0' を省略して記述することもできます。指数表現も可能です。また予約語として true (=1)と false(=0)を使うこともできます。

文字列はシングルクォーテーション(')で囲まれた範囲です。文字列中には改行コードとシングルクォーテーション自身を含めることはできません。このためどうしてもこれらの文字を入れたい場合は、0x0a などのように16進数表現を併用します。

説明文中の青色太字部分はその文字を記述することを表します。基本的にこれらの文字は大文字でも小文字でも構いません。斜体字はそれぞれの説明を参照してください。また[ ]で囲まれた部分は省略することができます。

7.3.2. 構文

[s*xx***f***yy* [*w*]] *Body*

アイテム	説明
xx	ストリーム番号。文字' s' 、' f' との間にはスペースを入れません。
yy	ファンクション番号。文字' f' 、' w' との間にはスペースを入れません。
w	ウェイトビット。指定する場合は' w' と記述します。省略可能です。
Body	メッセージのボディ。

ストリーム、ファンクション、ウェイトビットはひとかたまりで認識するため、これらの間にスペースや改行コードを入れないようにします。またストリーム、ファンクションを全て省略してメッセージボディのみを記述することもできます。

7.3.3. メッセージボディ

メッセージのボディは階層構造になっています。

1. リスト

{[ ] [*NumOfItem*]] *Body***}**
<[ ] [*NumOfItem*]] *Body***>**

アイテム	説明
NumOfItem	リストの数です。SECSIM との互換性のためだけに用意されています。この数字は無視されます。
Body	メッセージのボディ。他のアイテムを並べることができます。

2. ASCII 文字列

<a [*Strings*] **>**

アイテム	説明
Strings	文字列です。

長い文字列は分割して記述することもできます。また直接文字コードを記述することもできます。例えば

```
<a 'ABC' 'DEF' '012' 0x33 '4' 53 54 '789'>
```

これは下記と同じです。

```
<a ‘ABCDEF0123456789’>
```

3. **マルチバイト文字列**

マルチバイト文字列は通常の ASCII 文字列と同じように扱われます。しかし SEMI スタンダードにはこれを使ったメッセージは一つもありません。

```
<a2 [Strings]>
```

アイテム	説明
Strings	2 バイト文字列です。現在のバージョンでは DBCS にのみ対応しています。

4. **JIS8 文字列**

JIS8 文字列は半角カタカナを扱うためのものでした。しかし SEMI スタンダードにはこれを使ったメッセージは一つもありません。

```
<j [Strings]>
```

アイテム	説明
Strings	JIS8 文字列です。

長い文字列は分割して記述することもできます。また直接文字コードを記述することもできます。例えば

```
<j ‘ABC’ ‘DEF’ ‘012’ 0x33 ‘4’ 53 54 ‘789’>
```

これは下記と同じです。

```
<j ‘ABCDEF0123456789’>
```

5. **整数**

```
<i1 [Numbers]>
<i2 [Numbers]>
<i4 [Numbers]>
<i8 [Numbers]>
<u1 [Numbers]>
<u2 [Numbers]>
<u4 [Numbers]>
```

<u8 [Numbers]>

アイテム	説明
Numbers	数値です。それぞれ以下の意味となります。

型	説明
i1	符号付き 1 バイト整数
i2	符号付き 2 バイト整数
i4	符号付き 4 バイト整数
i8	符号付き 8 バイト整数
u1	符号なし 1 バイト整数
u2	符号なし 2 バイト整数
u4	符号なし 4 バイト整数
u8	符号なし 8 バイト整数

いくつかの数字を並べて記述することもできます。この場合は配列となります。例えば

```
<i1 1 0x02 3>
```

のように記述することができます。

現在のバージョンでは i8 と u8 に巨大な値を入れることはできません。

6. **浮動小数点数**

<f4 [FNumbers]>
<f8 [FNumbers]>

アイテム	説明
FNumbers	浮動小数点数です。それぞれ以下の意味となります。

型	説明
f4	4 バイト浮動小数点数
f8	8 バイト浮動小数点数

例えば

```
<f4 0 1.0 3.14>
```

のように記述します。

7. **バイナリ**

<b [Numbers]>

アイテム	説明
Numbers	数値です。

例えば

```
<b 0xff 0x3e 255 0>
```

のように記述します。

8. **ブーリアン**

```
<bool [Numbers]>
<boolean [Numbers]>
```

アイテム	説明
Numbers	数値です。true または false のいずれかです。

例えば

```
<bool true false 1 0>
```

のように記述します。0 以外は true となります。

8. Lydian.IPlugin

8.1. プロパティ

8.1.1. lydian

Lydian object.

```
public LydianObj lydian;
```

8.2. メソッド

8.2.1. コンストラクタ

コンストラクタ。継承クラスを作る際には、オーバーライドする必要があります。

```
public IPlugin(LydianObj lydian);
```

引数

名前	説明
lydian	Lydian オブジェクト。

8.2.2. LogMsg

メッセージをログに表示します。

```
Protected void LogMsg(ISecsII msg);
```

引数

名前	説明
Msg	SECS-II メッセージ。

もし SECS-II メッセージ以外の何かをログに表示したい場合、log.Write()メソッドを使ってください。

8.2.3. OnReceived

このメソッドは、Lydian がメッセージを受信すると呼ばれます。

```
Public abstract void OnReceived();
```

継承クラスはこのメソッドを必ず実装する必要があります。さもないとクラスオブジェクトを生成できません。

8.2.4. Send

msgo プロパティにセットされたメッセージを送信します。もし一次メッセージが指定されている場合は、Lydian は SECS-II ヘッダを返信メッセージとして設定します。

```
protected void Send(byte[] primary = null);
```

引数

名前	説明
Primary	SECS-II 一時メッセージ。もし null の場合、**msgo** の SECS-II ヘッダは一次メッセージの返信メッセージとして初期化されます。

Lydian はメッセージを送信し、通信ログに表示します。

9. Lydian.IPlugin.LydianObj

9.1. プロパティ

9.1.1. comm

物理層。

```
public Dorian.ICommunication comm;
```

Dorian.ICommunication は下記のクラスの基底クラスです。

- Dorian.Hsms
- Dorian.Forms.Hsms
- Dorian.Forms.HsmsView
- Dorian.SecsI
- Dorian.Forms.SecsI
- Dorian.Forms.SecsIView

Lydian は Dorian.Forms.Hsms と Dorian.Forms.SecsI を使います。

9.1.2. DeviceID

SECS-II ヘッダのデバイス ID です。

```
public ushort DeviceID;
```

9.1.3. handle

Lydian フォームのハンドルです。

```
public IWin32Window handle;
```

9.1.4. Host

ホストまたは装置。

```
public bool Host;
```

9.1.5. Hsms

HSMS または SECS-I。

```
public bool Hsms;
```

9.1.6. log

通信ログ。

```
public Dorian.ILog log;
```

Dorian.ILog は下記のクラスの基底クラスです。

- Dorian.Log
- Dorian.Forms.Log

- Dorian.Forms.LogView

Lydian は Dorian.Forms.LogView を使います。

9.1.7. msgi

受信メッセージ。

```
public Dorian.ISecsII msgi;
```

Dorian.ISecsII は下記のクラスの基底クラスです。

- Dorian.SecsII
- Dorian.Forms.SecsII

Lydian は Dorian.Forms.SecsII を使います。

9.1.8. msgo

送信メッセージ。

```
public Dorian.ISecsII msgo;
```

9.1.9. SystemBytes

SECS-II のシステムバイト。

```
public uint SystemBytes;
```

システムバイトは SECS-II メッセージを一意に識別するために使われ、送信するたびにインクリメントする必要があります。