Lydian Simulator

Manual

Jazz Soft, Inc.

1. Revision History

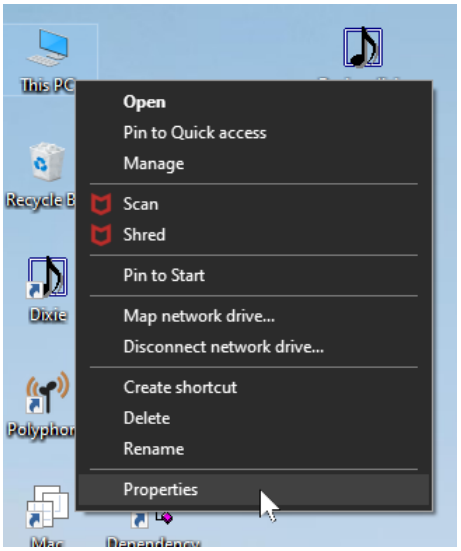
Version	Date	Name	Description
1.00	Dec, 5th, 2021	Carl Okada	Created
1.00a	Dec, 7th, 2021	Carl Okada	Added Dorian PATH.
1.01	Dec, 7th, 2021	Carl Okada	Changed dependency injection.
1.02	Feb, 19th, 2022	Carl Okada	Added auto reply.

2. Table of Contents

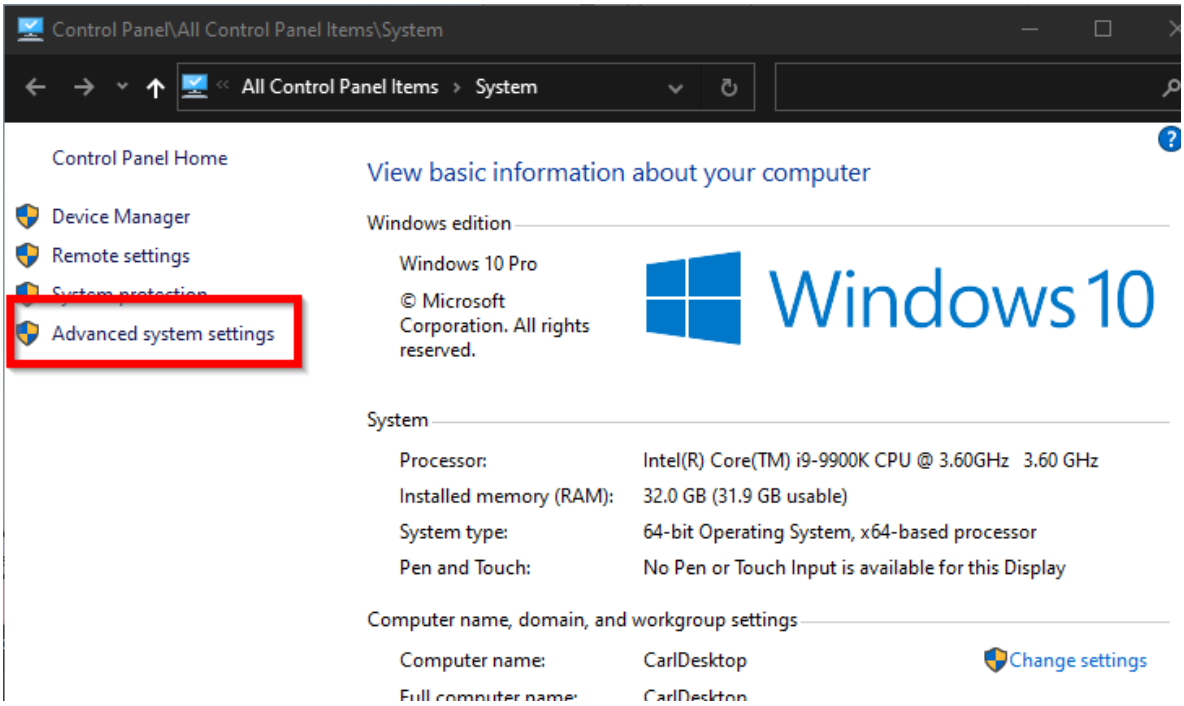
3. Install Dorian

It is needed to add Dorian installation folder to PATH.

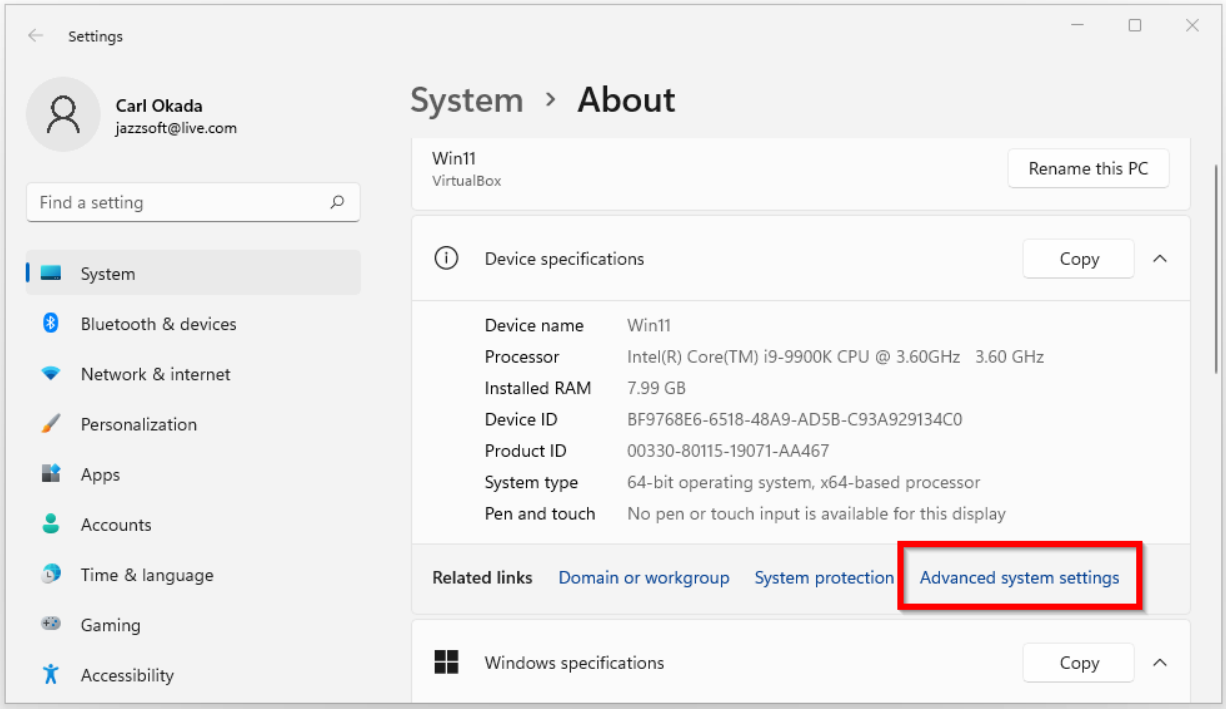
1 Right-click **This PC** on the desktop and click **Properties**.



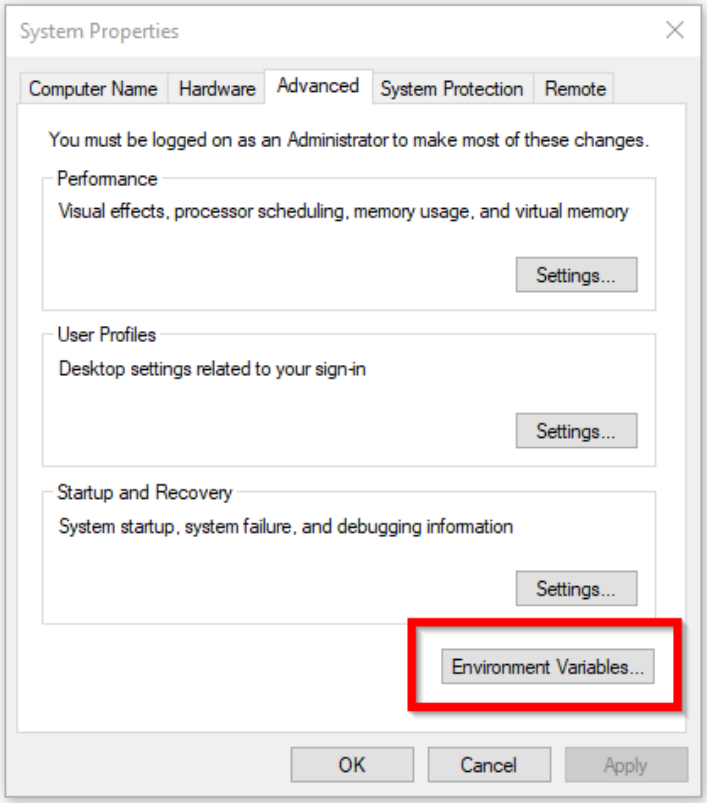
2 Click Advanced system settings.



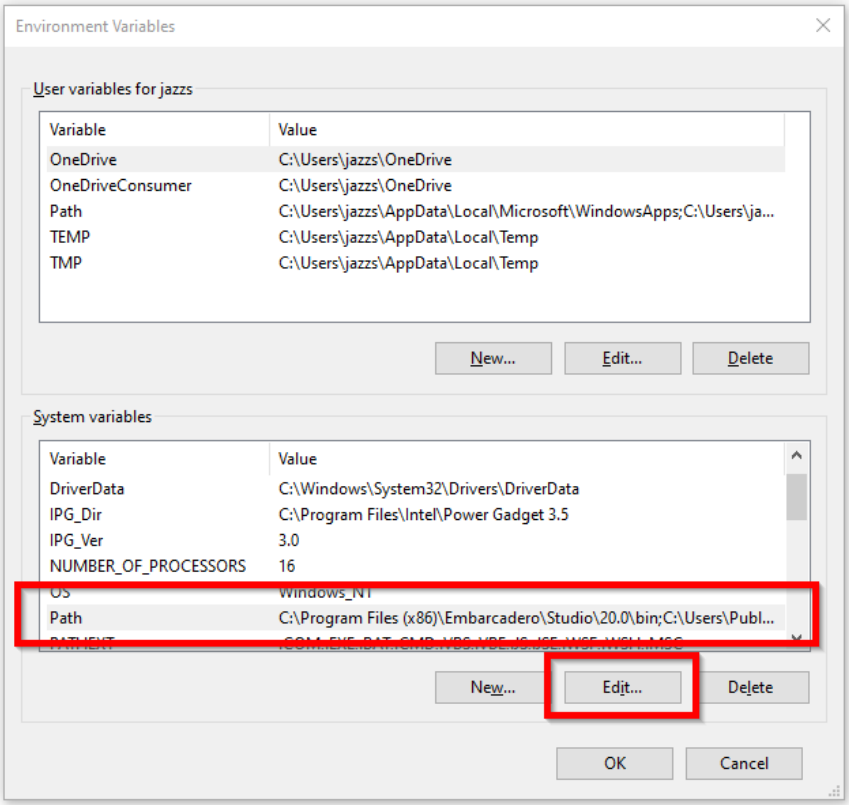
Windows 11 would be in the middle of the screen.



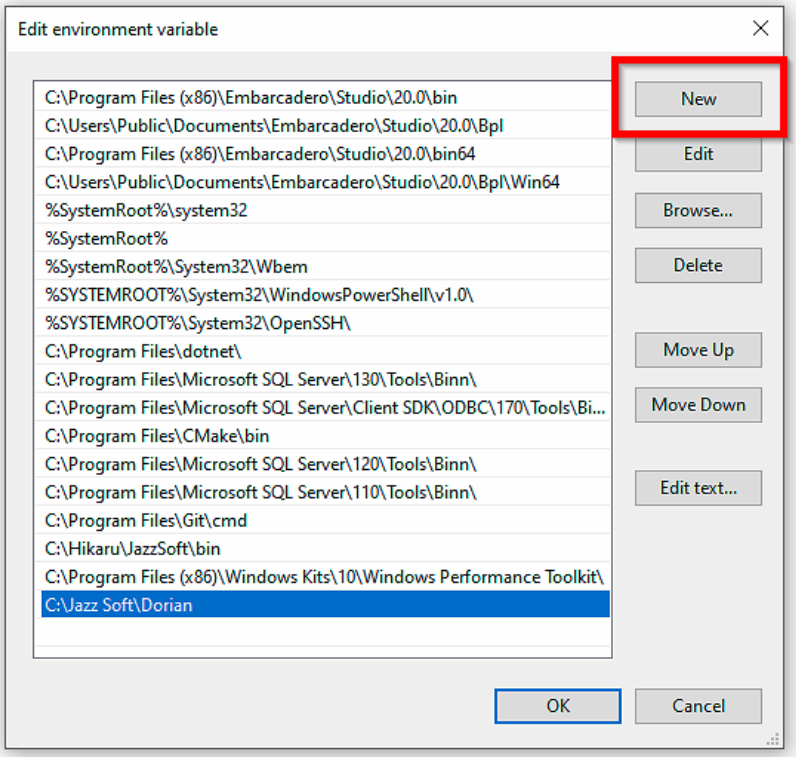
3 Click **Environment Variables** button.



4 Select **Path** in **System variables** and click **Edit** button below.

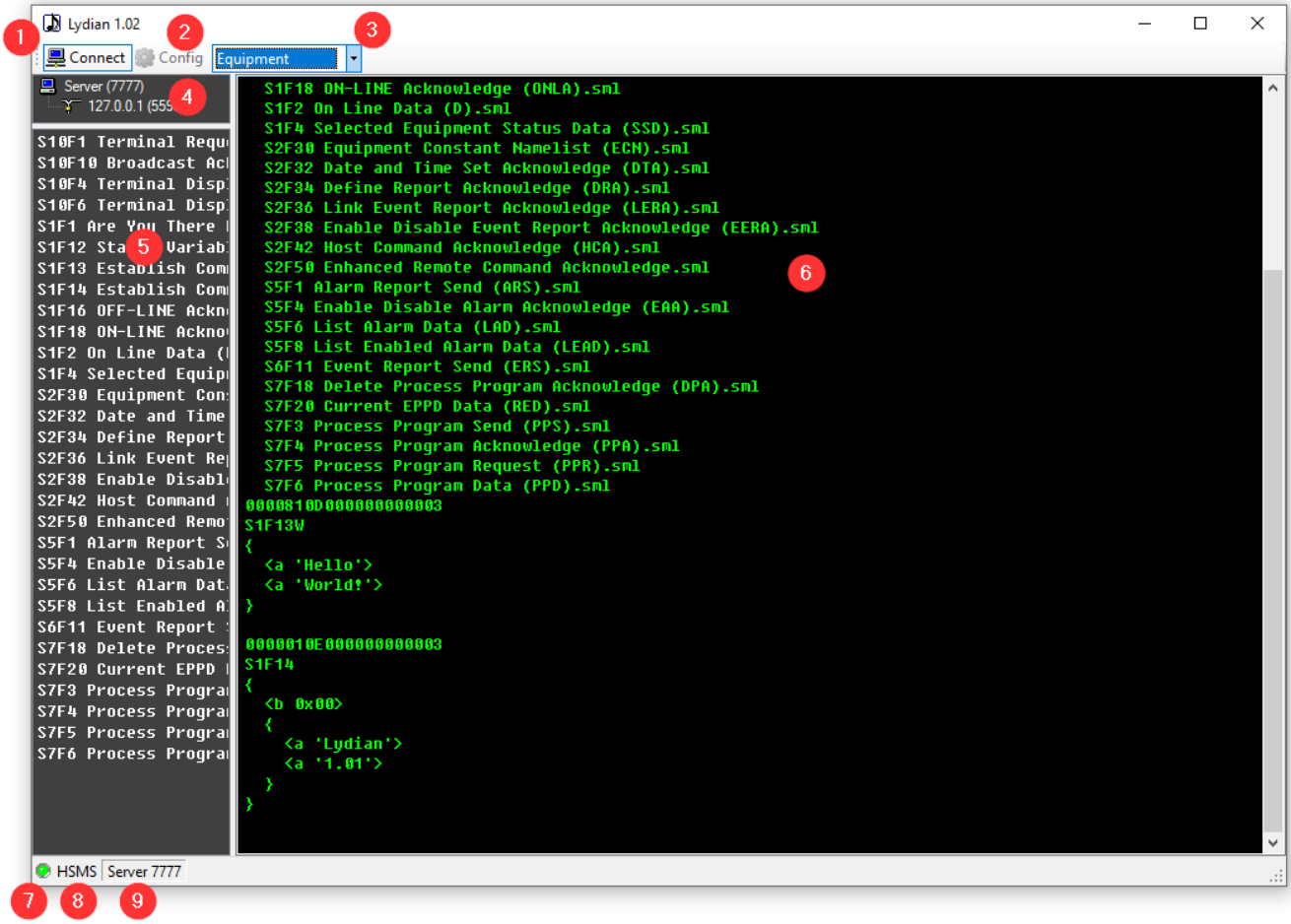


5 Click **New** button and type in the assembly locations.



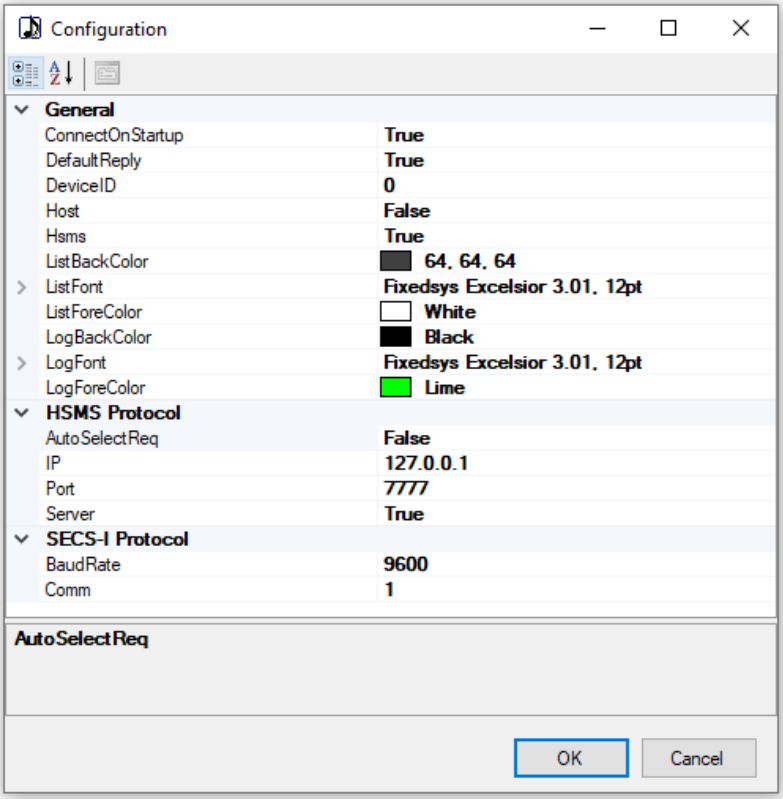
4. User Interface

4.1. Main Screen



Item	Description
1	**Connect button** enables or disables the physical layer. Physical layer would be either HSMS or SECS-I. If enabled, this button shows with blue box (“checked”). For HSMS passive entity (server), it opens specified TCP port and listens to incoming connection from active entity (client).
2	**Config button** shows the settings dialog box. It is disabled when it is connected.
3	**SECS-II message sets** are listed here.
4	**Communication status** is displayed here.
5	**SECS-II messages** are listed here. User can double-click the message and Lydian sends out the message.
6	**Communication log** is displayed here.
7	**Connection indicator** shows green if connected, and gray if not connected. This is same as the blue box around Connect button.
8	**Physical layer** would be either HSMS or SECS-I.
9	**Connection parameters** are displayed here. HSMS would be “Server” (passive entity) or “Client” (active entity) followed by TCP port number. SECS-I would be serial port number and baud rate.

4.2. Configuration Dialog Box



4.2.1. General

Item	Description
ConnectOnStartup	Connects when Lydian app starts up.
DefaultReply	Return <b 0> if script doesn't reply and also reply message is not defined.
DeviceID	Device ID of SECS-II header.
Host	Lydian acts like host or equipment.
Hsms	Physical layer is HSMS or SECS-I.
ListBackColor	Background color of the SECS-II message list.
ListFont	Font of the SECS-II message list.
ListForeColor	Foreground color of the SECS-II message list.
LogBackColor	Background color of the log console.
LogFont	Font of the log console. It is highly recommended to use a fixed-pitch font.
LogForeColor	Foreground color of the log console.

4.2.2. HSMS Protocol

Item	Description
IP	IP address. This has to be 127.0.0.1 for passive entity.
Port	TCP port number.
Server	HSMS passive entity (server) or active entity (client).

4.2.3. SECS-I Protocol

Item	Description
BaudRate	Baud rate (communication speed). Typical value is 9600 bps.
Comm	Serial port number.

5. Folder Structure

5.1. Folders

When you configure each simulation settings, the easiest way is to save each simulation in its own folder. Other simulation settings will not affect at all.

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

Folder	Description
Lydian	The folder where Lydian.exe and settings are placed.
msgs	Messages.
plugins	Plugins.

If you want to create your own message set, make a sub-folder under msgs/ folder.

5.2. Files

5.2.1. Lydian Folder

Following files will be placed or generated.

File	Description
debug.log	Log file.
debug001.log	Log backup file.
Lydian.config	Lydian settings.
Lydian.exe	Lydian simulator executable file.

5.2.2. msgs Folder

Message files have to be stored in this folder. The file extension has to be "**.sml**".

5.2.3. plugins Folder

Plugin files have to be stored in this folder. The file extension has to be "**.dll**".

6. Plugin

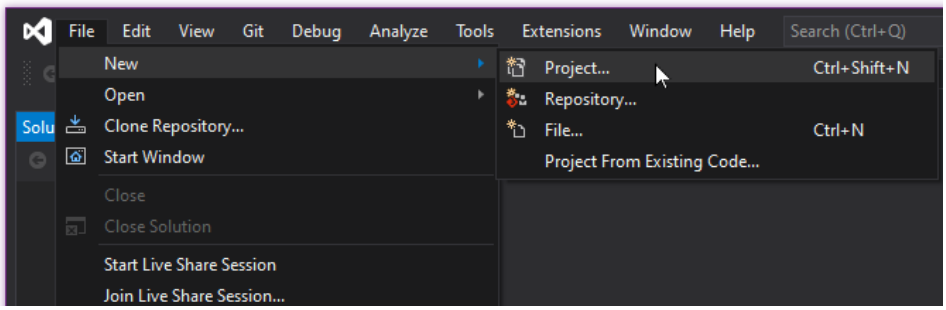
Scripts are created as plugin. Lydian can import multiple plugins. Plugins can be created C#, Visual Basic or any other .NET compliant languages.

6.1. C# Tutorial

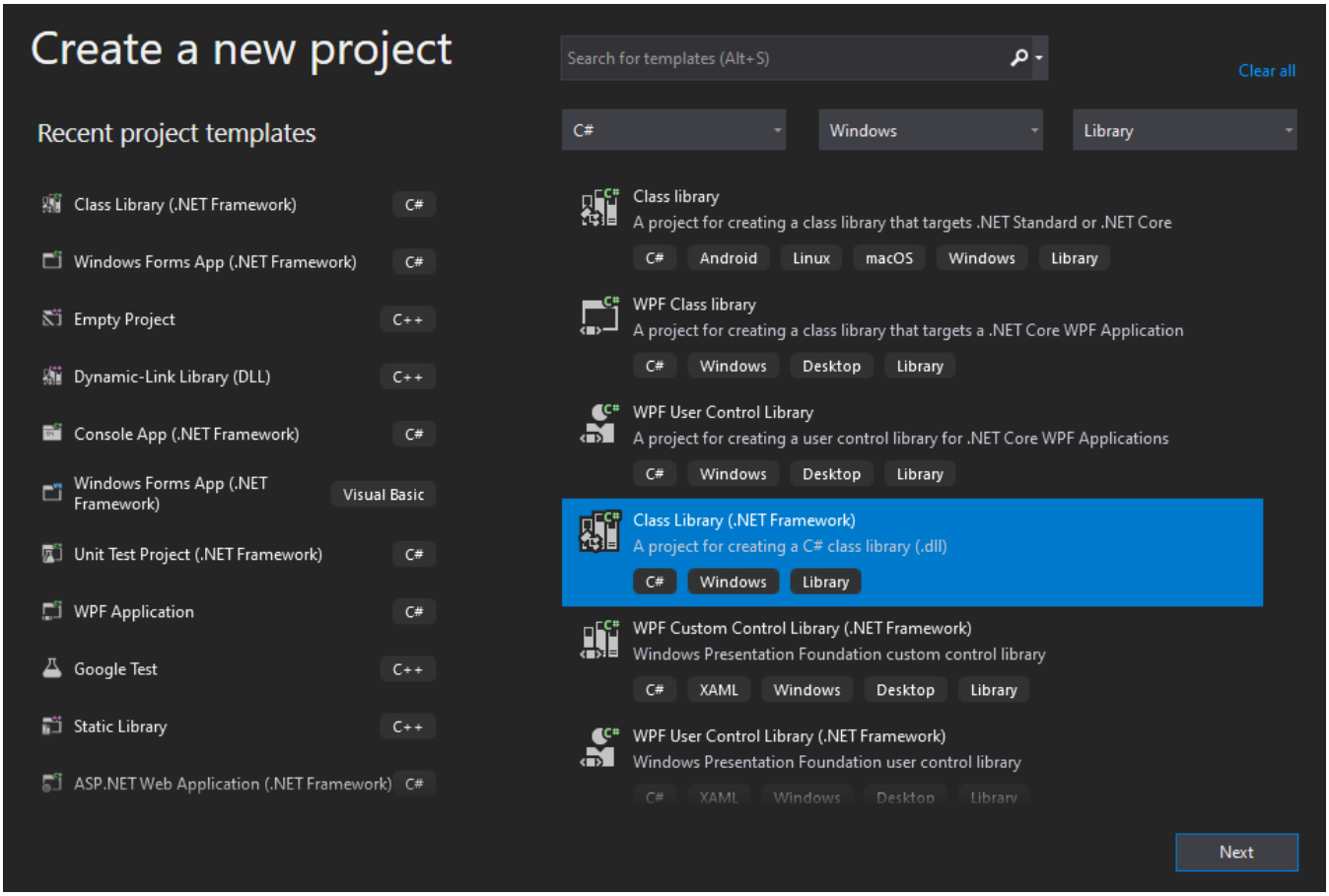
Let's make a plugin with Visual Studio 2019 running on Windows 10. This procedure is same for Visual Studio 2022 on Windows 11 or other combinations of previous versions of Visual Studio and Windows.

6.1.1. Create Your Plugin Project

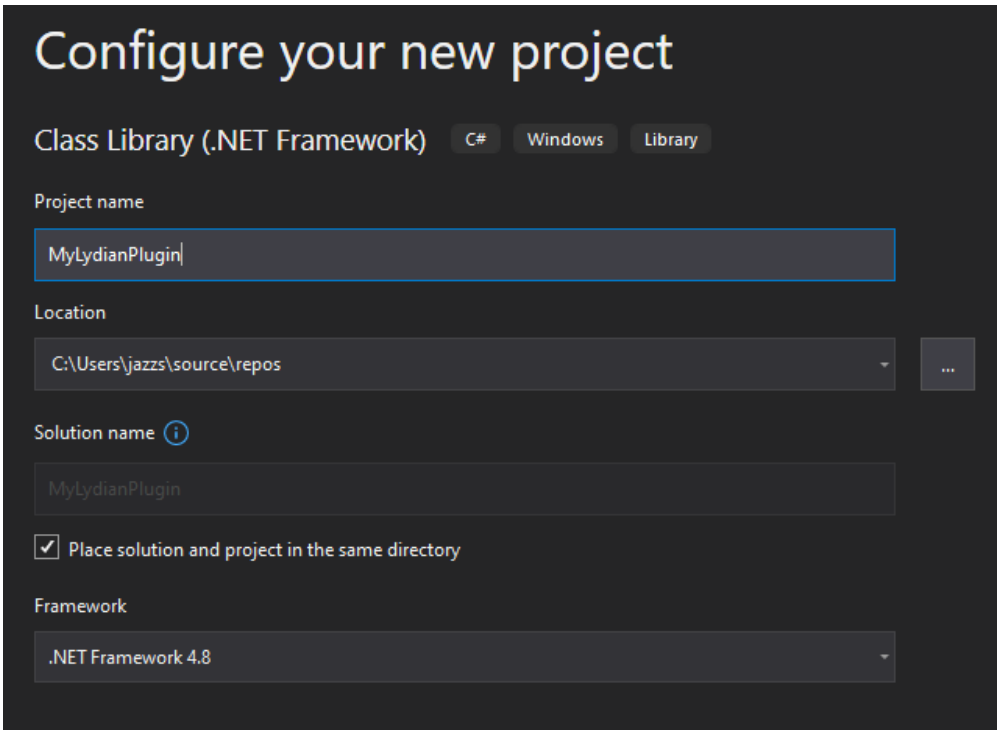
6 Create a new project from [**File**] – [**New**] – [**Project...**] on the menu.



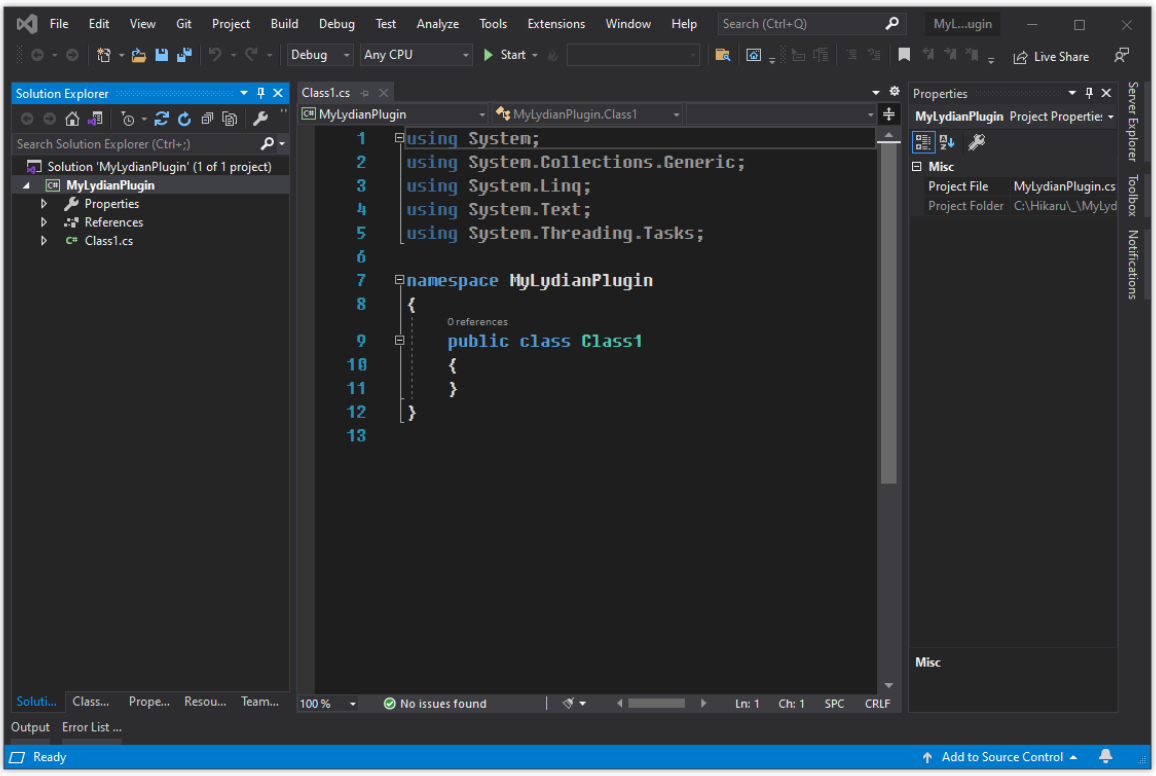
7 Select C# – Windows – Library in the dropdown lists. Choose **Class Library (.NET Framework)** project. Be careful there are very similar project types.



8 Give your favorite name to the project, choose the location of the project and choose .NET Framework version. I prefer "Place solution and project in the same directory" so that everything is in the same folder, instead of nested folders. And then press Create button.



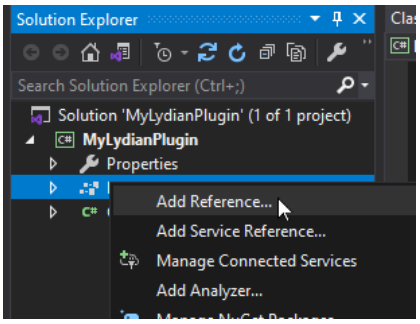
9 Your plugin project is created.



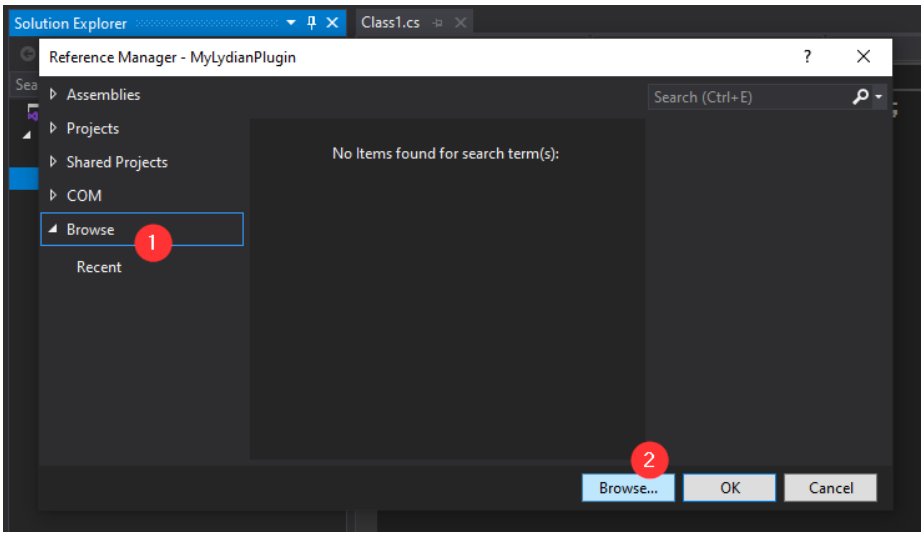
6.1.2. **Add Dorian Assembly**

Lydian uses Dorian. And Dorian.NET.dll contains an abstract class for Lydian plugin.

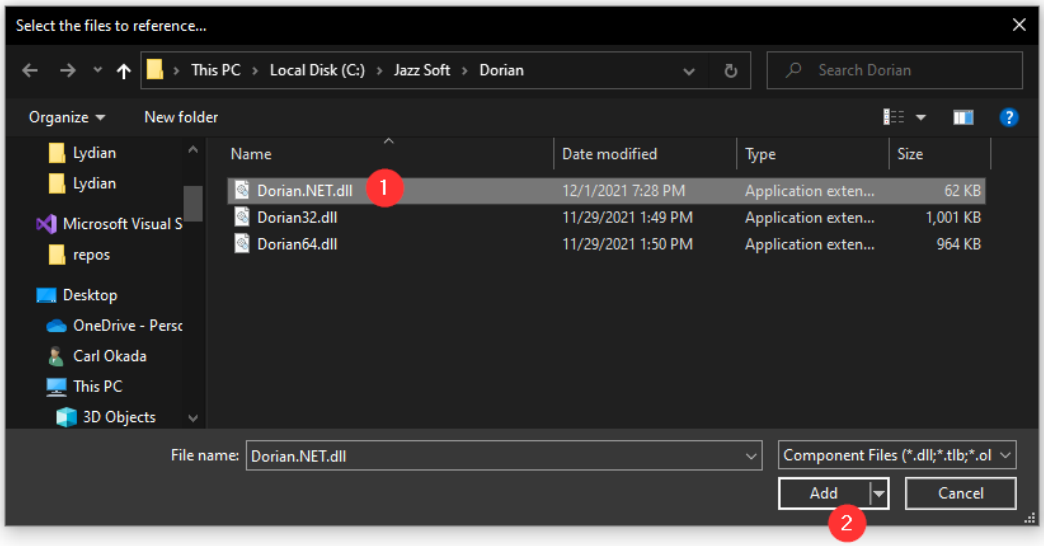
1 Go to Solution Explorer and right-click on **References**. And then choose **Add Reference...**.



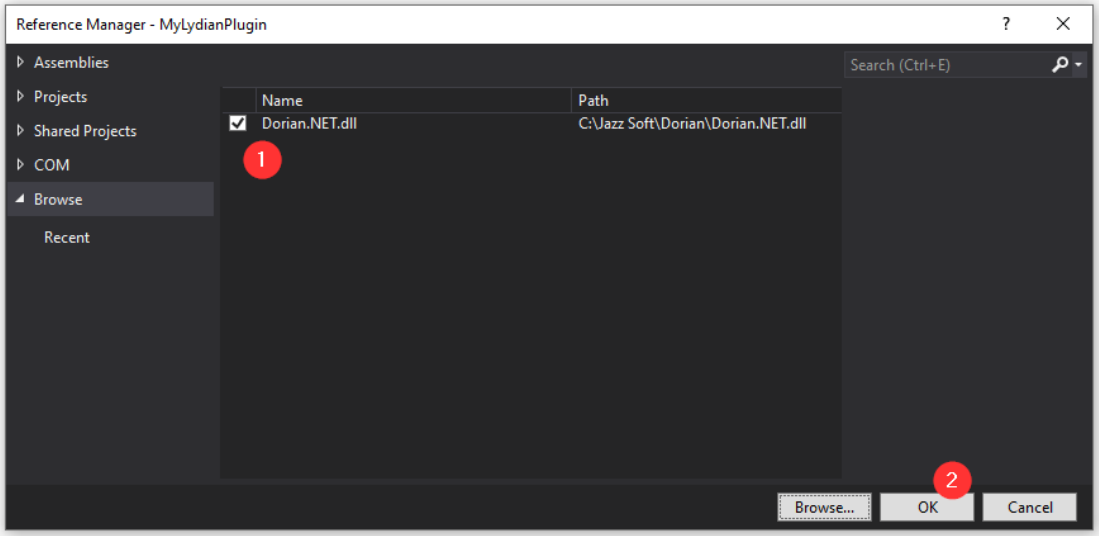
2 Select **Browse** on the left pane and click **Browse...** button.



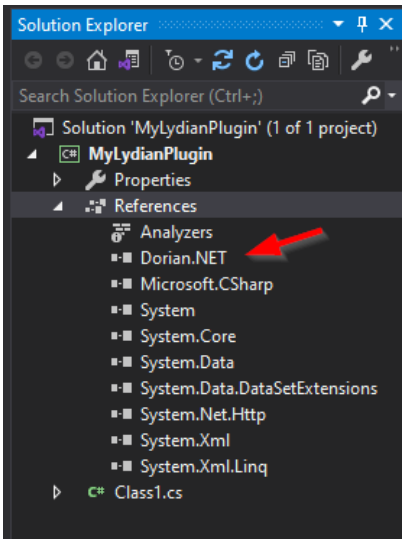
3 Go to **C:\Jazz Soft\Dorian** folder and select **Dorian.NET.dll**. And then click **Add** button.



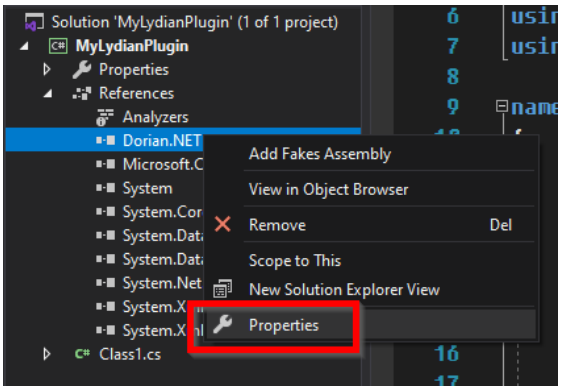
4 Double-check to make sure **Dorian.NET.dll** is checked. And then press OK button.



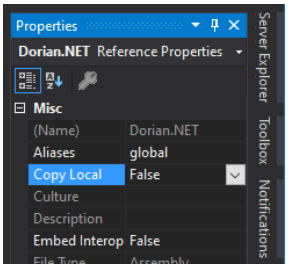
5 Dorian.NET should be added to the References.



6 Right-click on it and choose **Properties**.

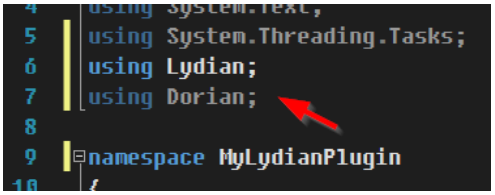


7 Change **Copy Local** to **False**. We don't need to copy it.

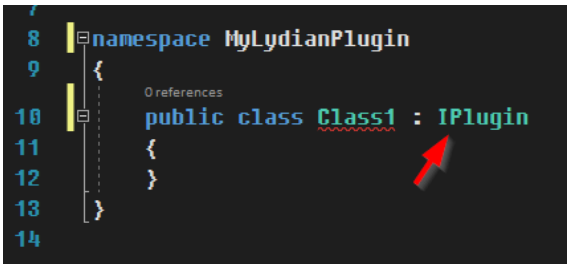


6.1.3. Inherit from Lydian.Plugin

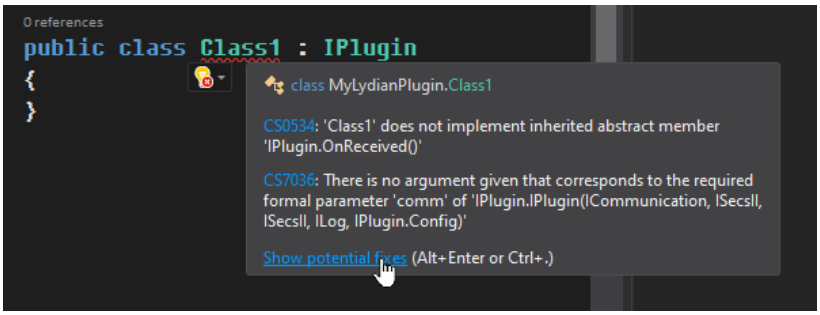
1 Add "**using Lydian;**" and "**using Dorian;**" for your convenience. This is optional.



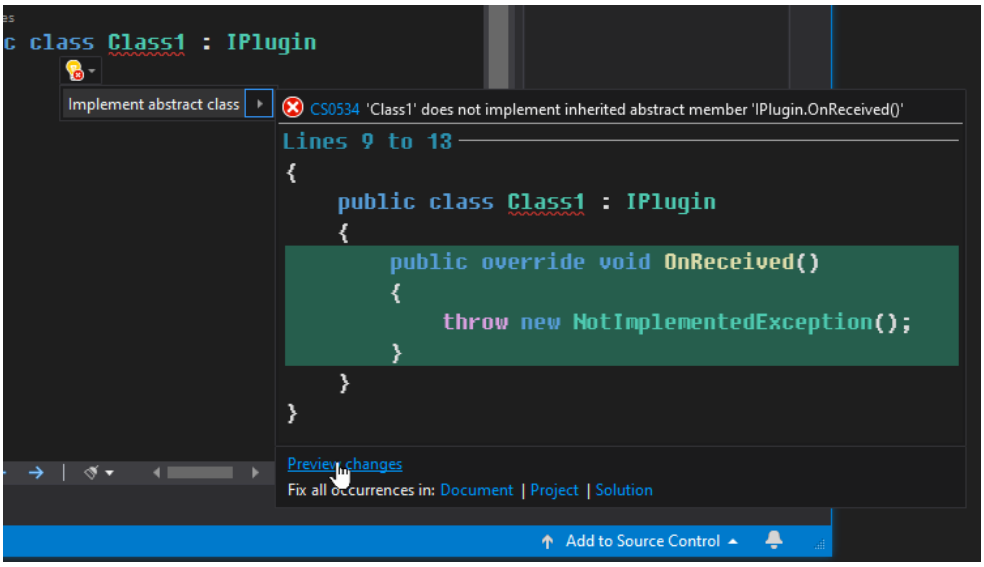
2 Make Class1 class inherit from **Lydian.IPlugin**.



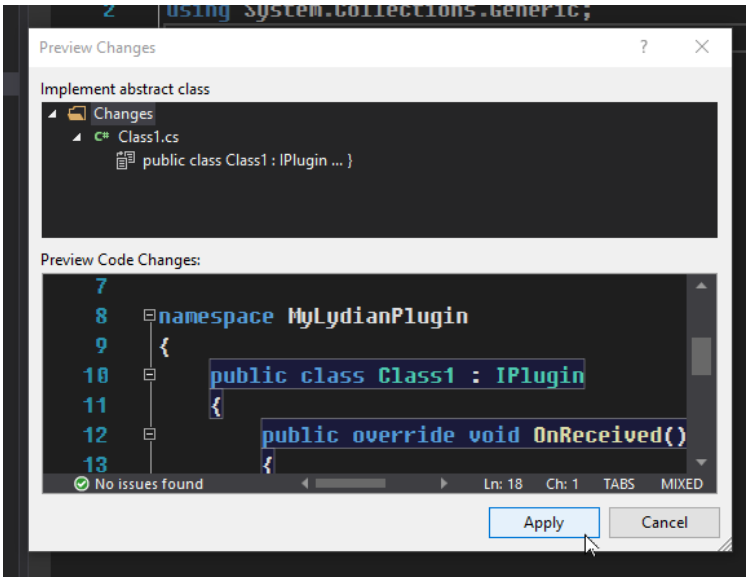
3 Lydian.IPlugin is an abstract class. You have to implement the necessary function. That's why you see red underline. Hover the mouse over the class name. And click **Show potential fixes**.



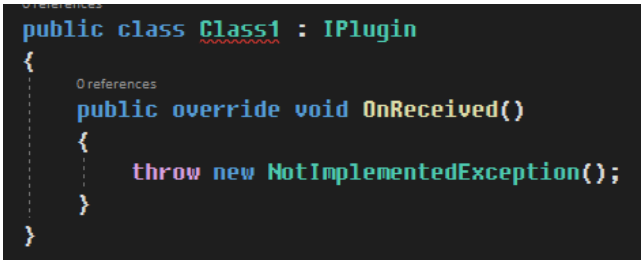
4 Click **Preview changes**.



5 Click **Apply** button.



6 **OnReceived**() function has been overridden but you still see red line under the class name, because Lydian.IPlugin requires specific constructor for the dependency injection (DI).



7 Add following constructor code to your class. This should resolve the error.

```
public Class1(LydianObj lydian)
    : base(lydian)
```

```
{
}
```

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

6.1.4. Add Info

1 You can directly write to the message log window by using the **log** object. These arguments will be copied to your local property in the base class (IPlugin abstract class) so that you can access any time.

```
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. Reply Message

OnReceived() will be called when Lydian receives a message. Let's respond to S1F13.

1 Incoming message is set to **msgi** property in Lydian.IPlugin class. Since your class is inherited from Lydian.IPlugin, you can access **msgi** property.

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

2 Outgoing message should be set to **msgo** property. To send a message you can use **Send**() method also defined in Lydian.IPlugin.

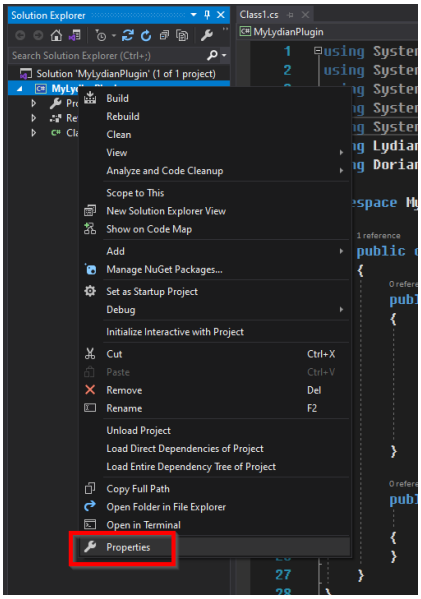
```
    lydian.msgo.Sml = "{<b 0>{}}";
    Send(lydian.msgi.Msg);
```

3 OnReceived() method is like this.

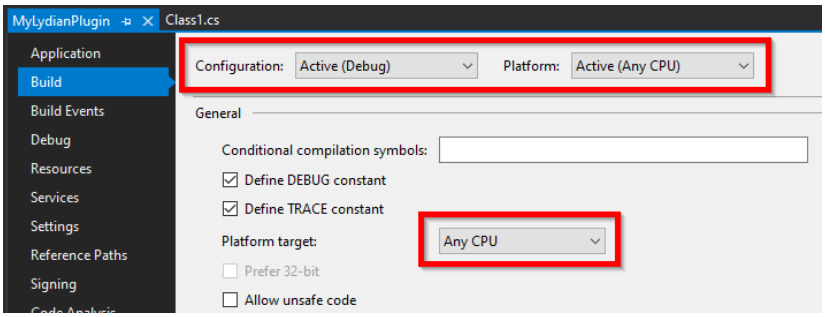
```
0 references | Carl Okada, 6 hours ago | 1 author, 1 change
public override void OnReceived()
{
    if (lydian.msgi.Stream == 1 && lydian.msgi.Function == 13)
    {
        // S1F13
        lydian.msgo.Sml = "{<b 0>{}}";
        Send(lydian.msgi.Msg);
    }
}
```

6.1.6. Setup Debug Environment

1 As explained above, the plugin DLL has to be stored in “plugin” folder. Let’s change the DLL output folder on Visual Studio. Right-click on the project and click **Properties**.

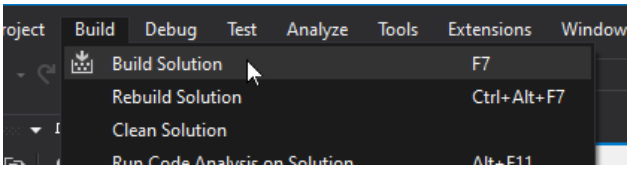


2 Choose **Build** on the left pane. Check to make sure the Configuration is **Debug** and the Platform is **Any CPU**. And the Platform target is set to **Any CPU**.

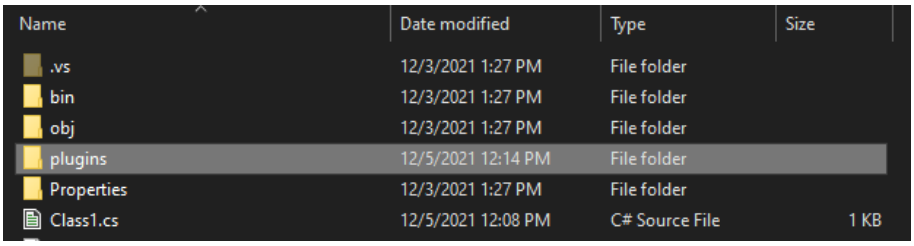


3 On the same page, change the Output path to **plugins**.

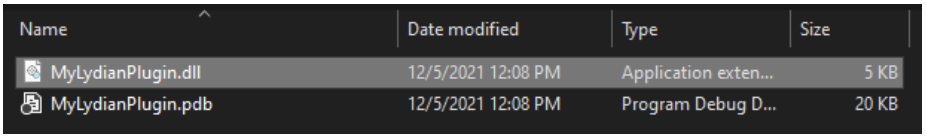
4 Build your project. Click [**Build**] – [**Build Solution**] on the menu.



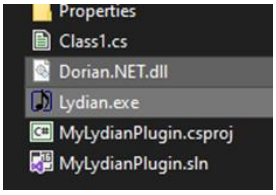
5 Now you will see **plugins** folder is created.



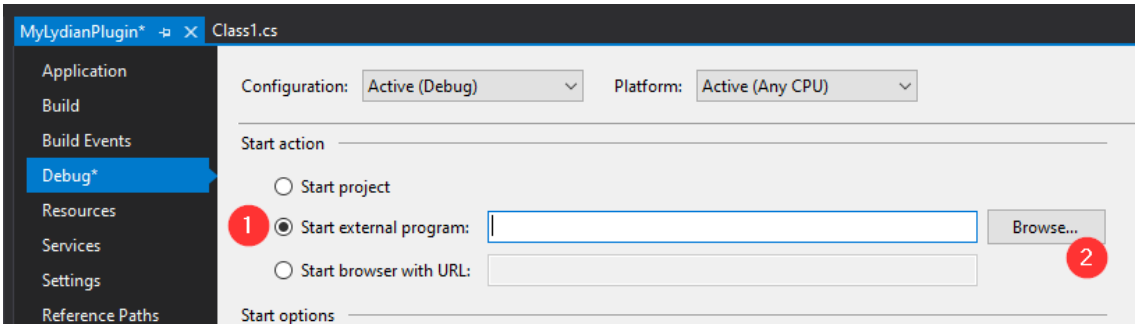
6 In the **plugins** folder, you should see your plugin DLL.



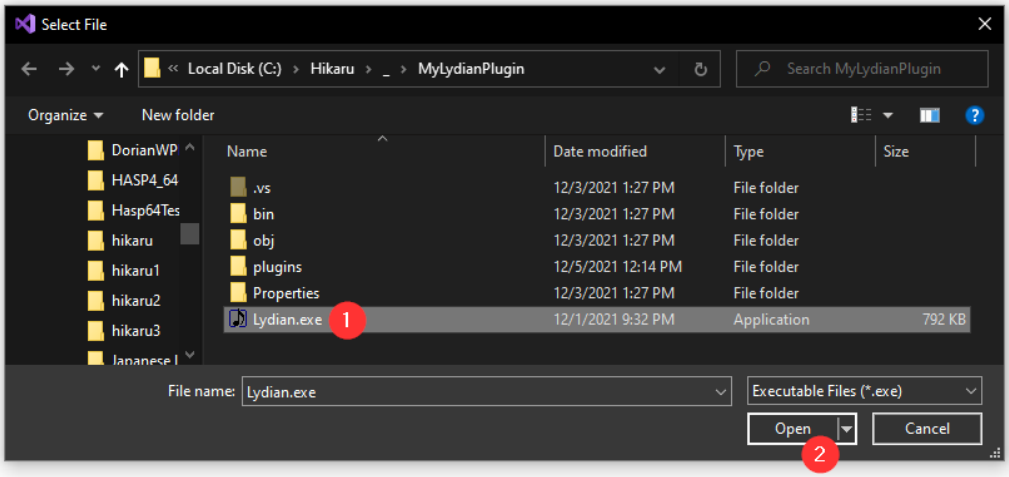
7 Go to **C:\Jazz Soft\Lydian** folder and copy **Lydian.exe** and **Dorian.NET.dll** to your project folder.



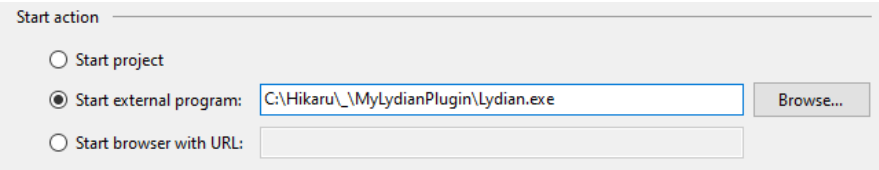
8 Go back to the project's property again and choose **Debug** on the left pane. Select **Start external program** and click **Browse...** button.



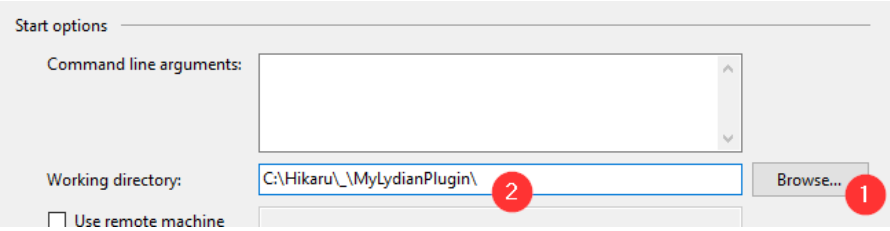
9 Choose **Lydian.exe** and click **Open** button.



10 Now Lydian.exe is selected for debugging.

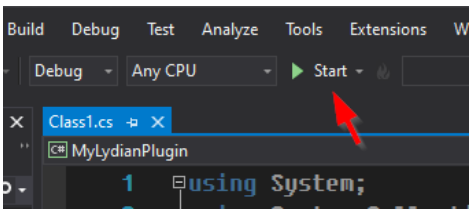


11 On the same page, do the same thing for **Working directory**. This has to be the same folder as Lydian.exe.



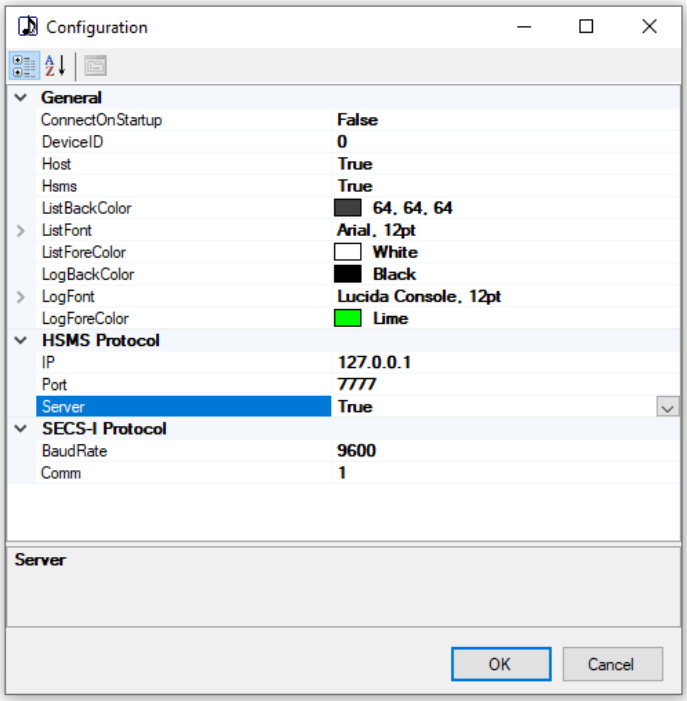
6.1.7. Debug Plugin with Lydian

1 Check to make sure **Debug** and **Any CPU** is selected and click **Start** button on Visual Studio.

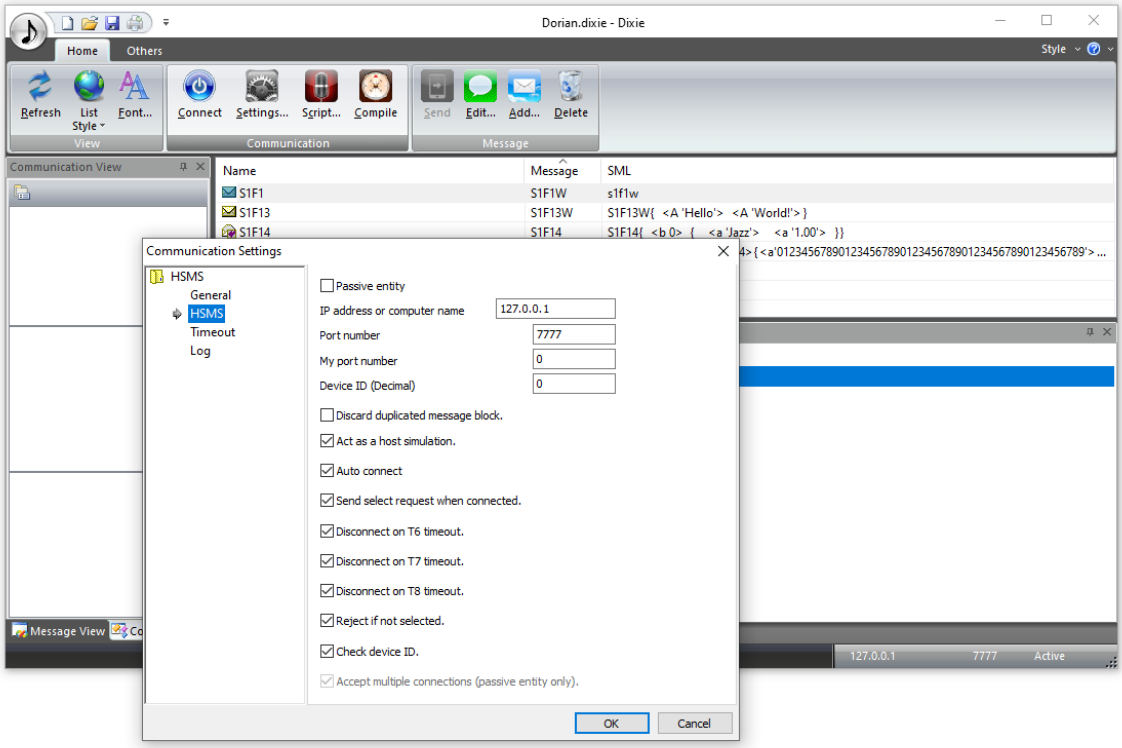


2 Lydian should launch but communication settings are not configured yet. Click **Config** button.

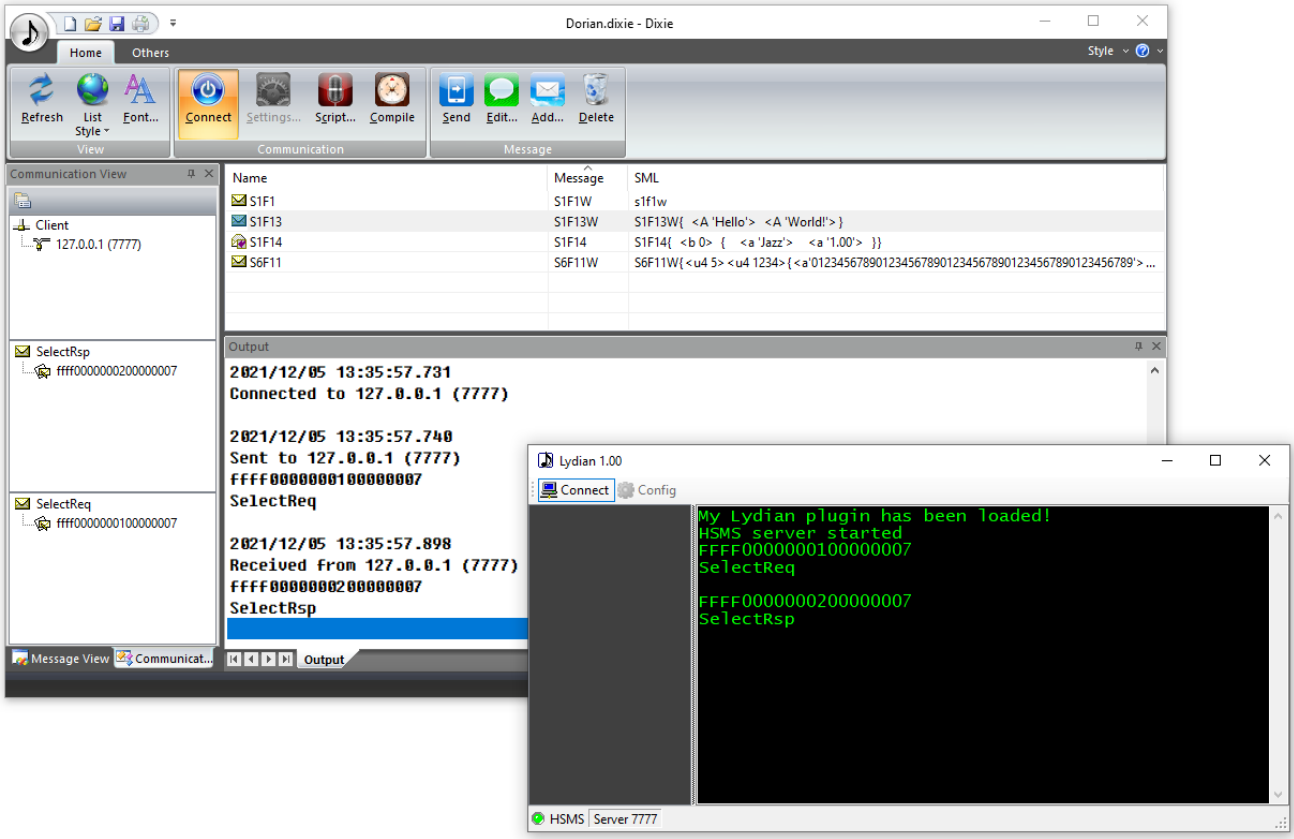
3 Configure the parameters. In this tutorial, we will use HSMS server and the port number is 7777.



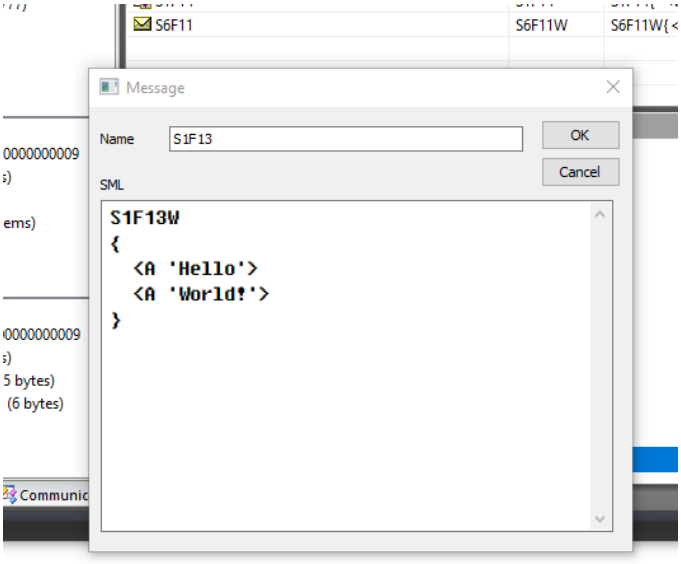
4 Launch Dixie simulator (or your favorite HSMS simulator app) and configure it as HSMS client.



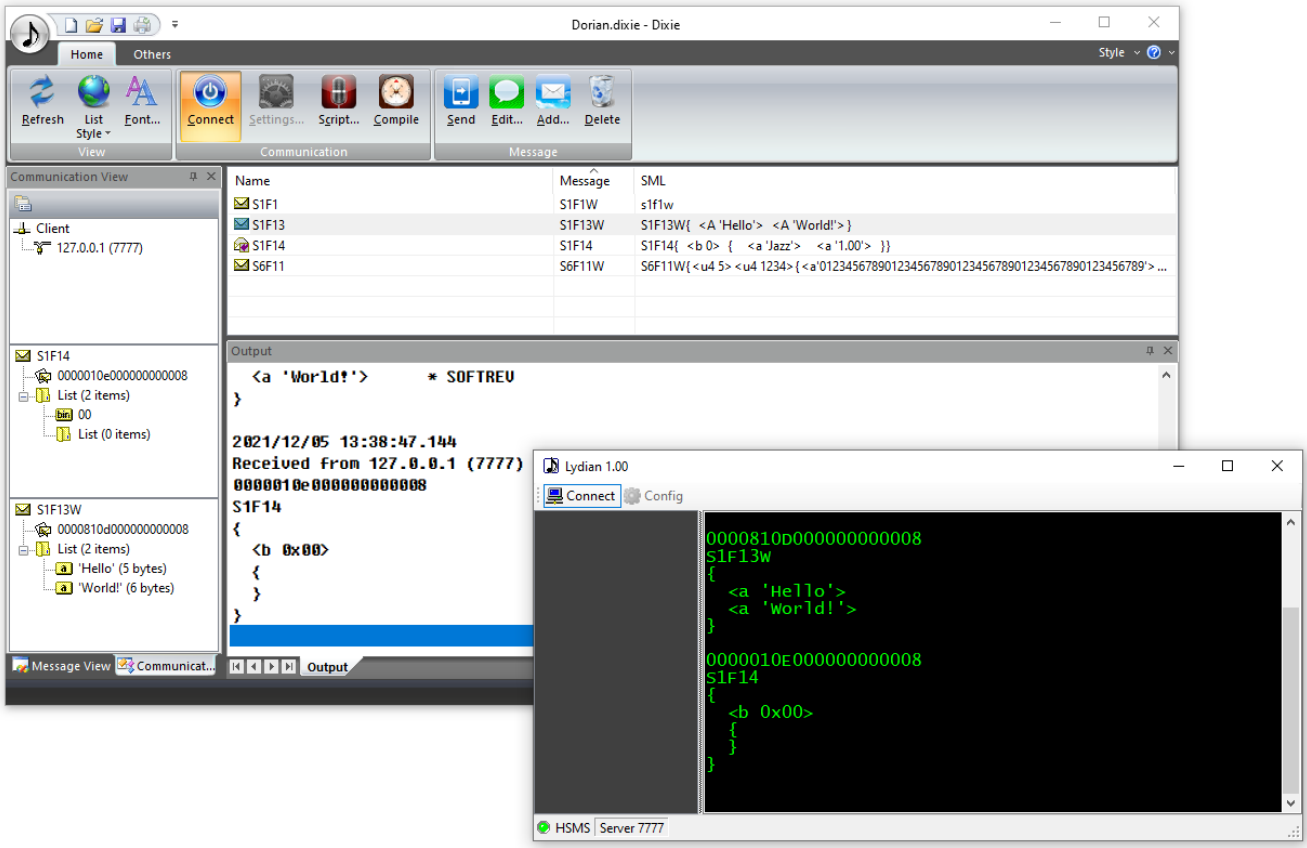
5 Click Connect button. It should start communicating.



6 S1F13 should be like this.



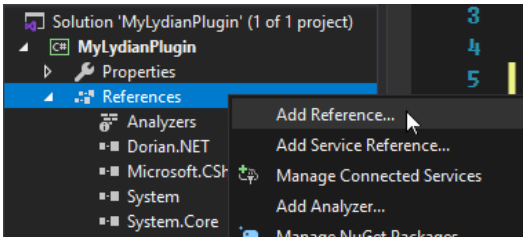
7 When sending S1F13, your plugin replies S1F14. You can set a breakpoint in the source code if you want.



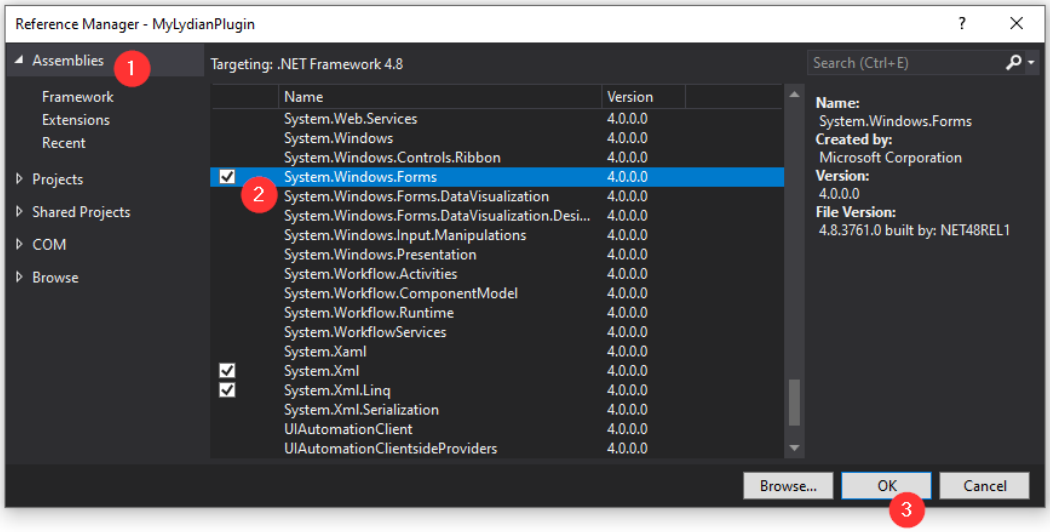
6.2. Using Timer

6.2.1. **Send S1F1 Periodically**

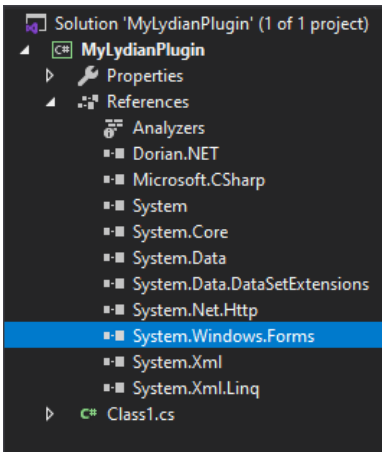
1 There are several different timers in .NET libraries but only **System.Windows.Forms.Timer** works without dealing with multi-threading loophole. Right-click on **References** and choose **Add Reference...**.



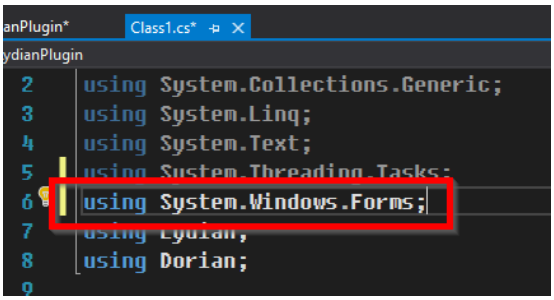
2 Choose **Assemblies** on the left pane and check **System.Windows.Forms**.



3 System.Windows.Forms has been added.



4 Add "**using System.Windows.Forms;**" in the source code for your convenience. This is optional.



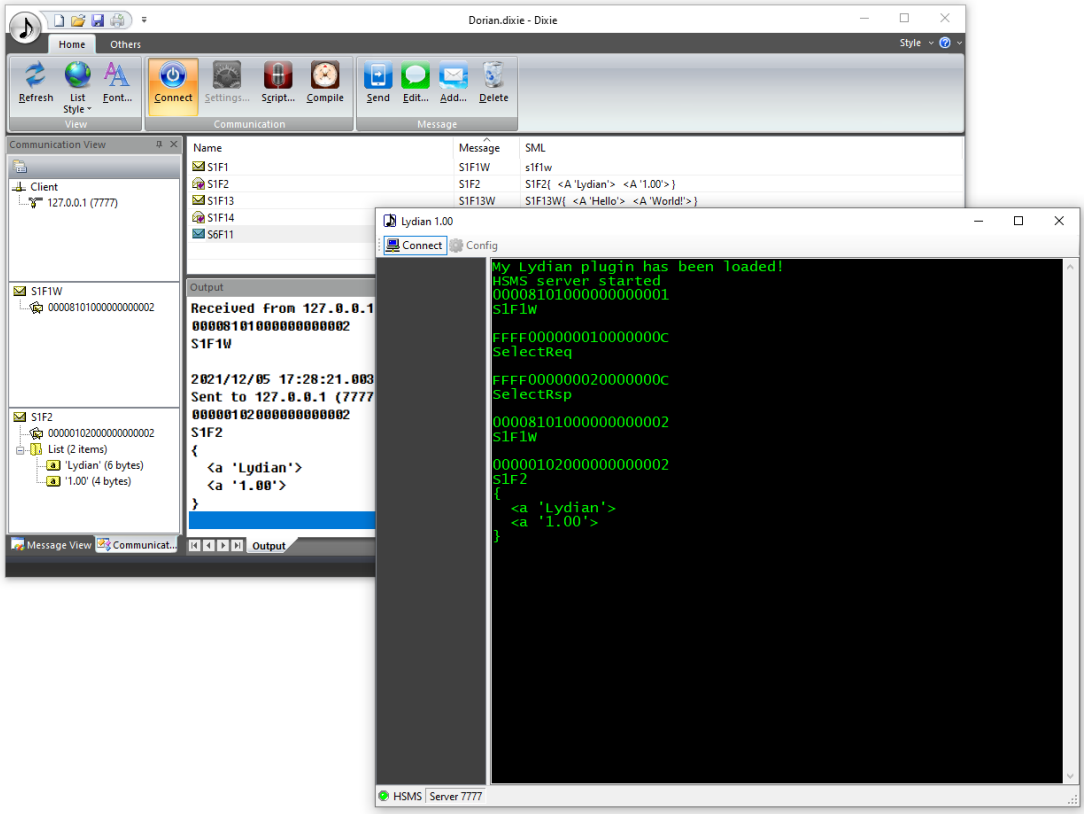
5 Create a timer object in the class. Timer interval is 10 seconds.

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

6 Let's use relatively new technique, lambda expression, so that we don't need to create a function that is only used for the timer.

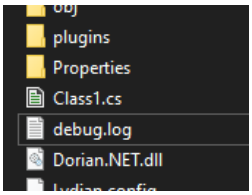
```
timer.Tick += (sender, e) =>
{
    if (!lydian.comm.Connect)
        return;

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

7 The plugin keeps sending S1F1 message every 10 seconds.

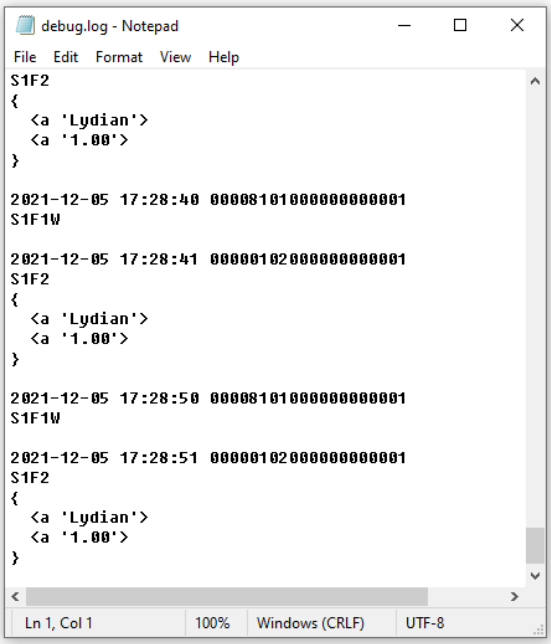


6.2.2. Log File

1 Log file is created as **debug.log**.



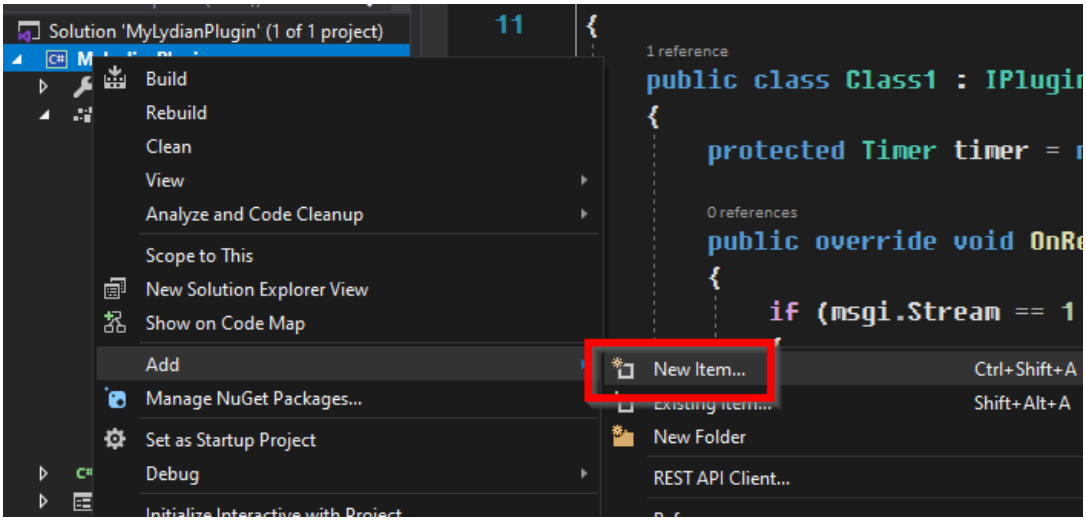
2 Date and timestamp were added.



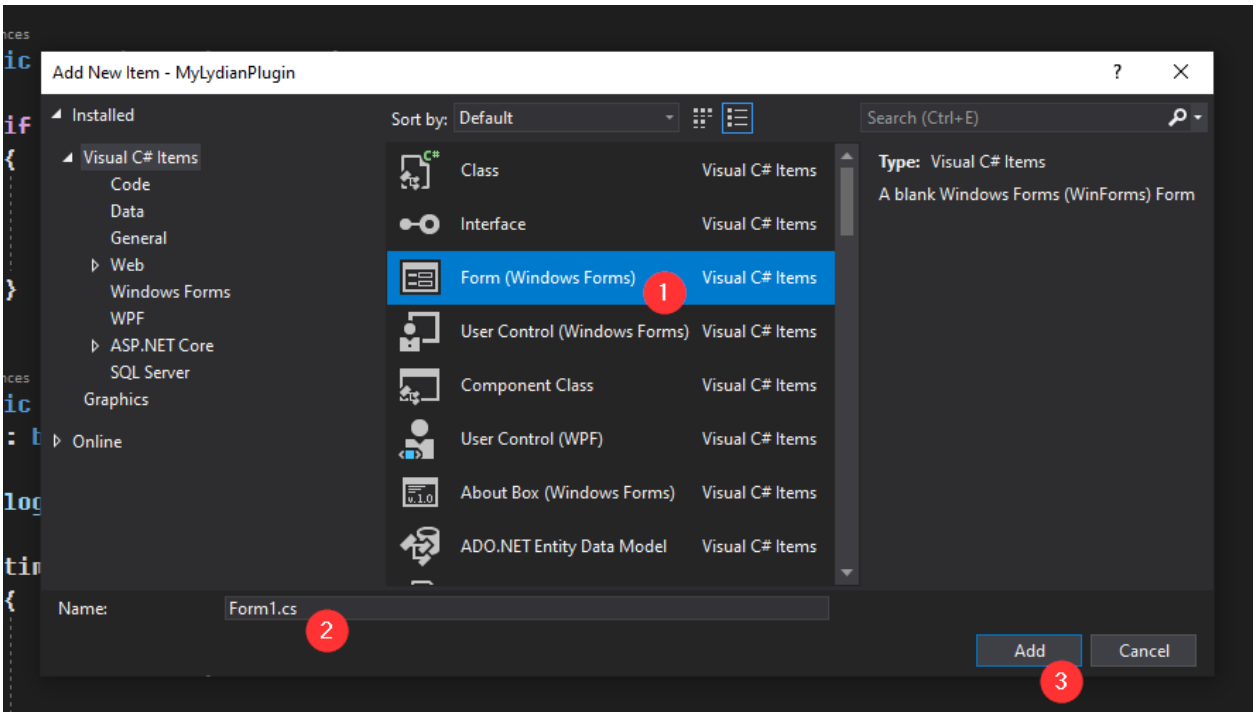
6.3. Add User Interface

6.3.1. Add Dialog Box

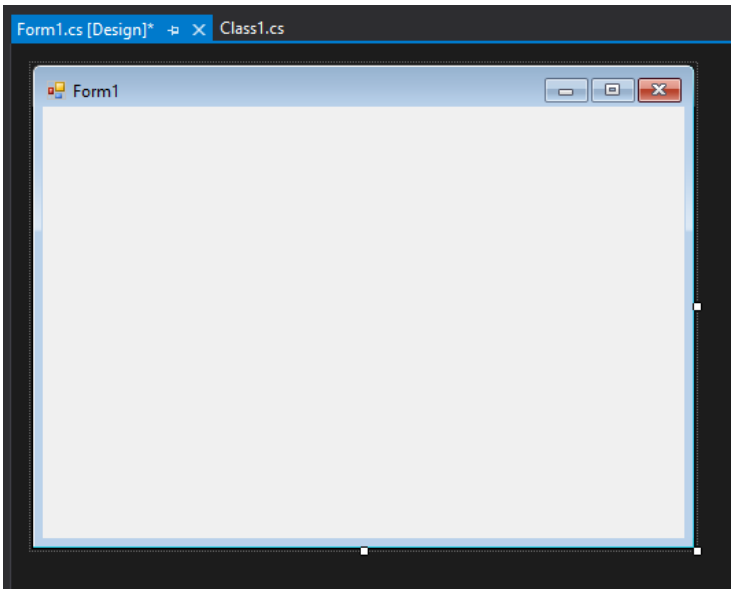
1 Right-click on the project and choose **Add** – **New Item...**.



2 Choose **Form (Windows Forms)**, type in your favorite name and click **Add** button.



3 A form is added to the project.



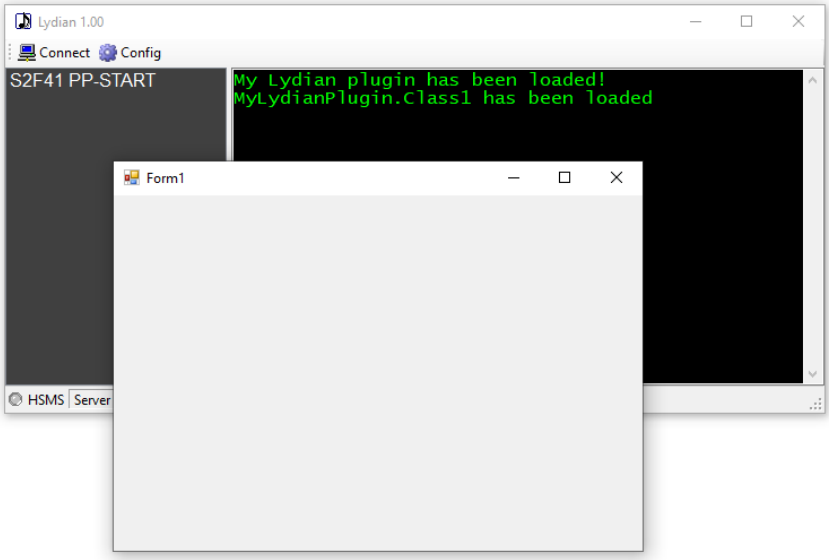
4 Create the form object.

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

5 Call Show() method of the form.

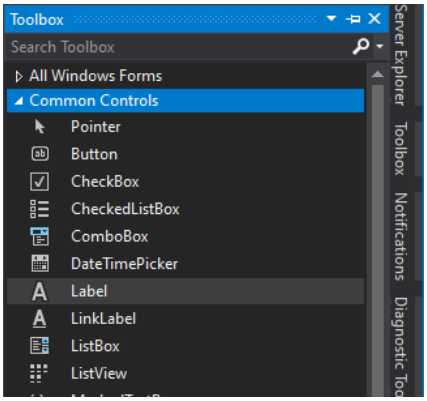
```
0 references | 0 changes | 0 authors, 0 changes
public Class1(LydianObj lydian)
    : base(lydian)
{
    lydian.log.Write("My Lydian plugin has been loaded!");

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

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

    form.Show();
}
```

6 A modeless dialog box will popup.

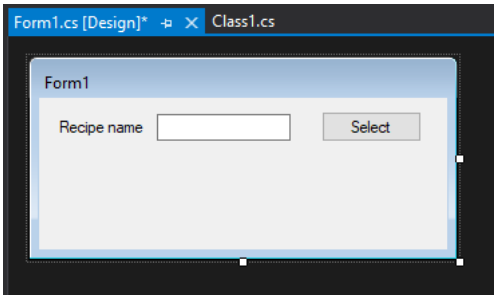


6.3.2. Select Recipe

1 Open Toolbox.



2 Place a label, text box and button. Hide the control box.



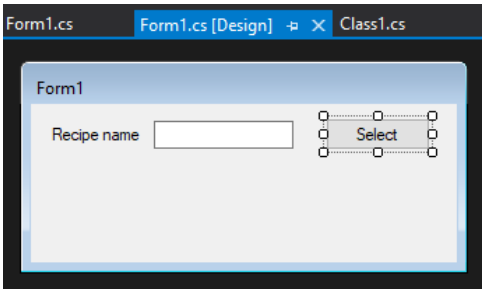
3 Prepare the reference of the Lydian.IPlugin objects in Form1.

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

    1 reference
    public Form1()
    {
```

4 Copy the reference from Class1 to Form1. Note that it's not cloning, just copying the references and the actual objects are in Lydian.

```
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);
}
```

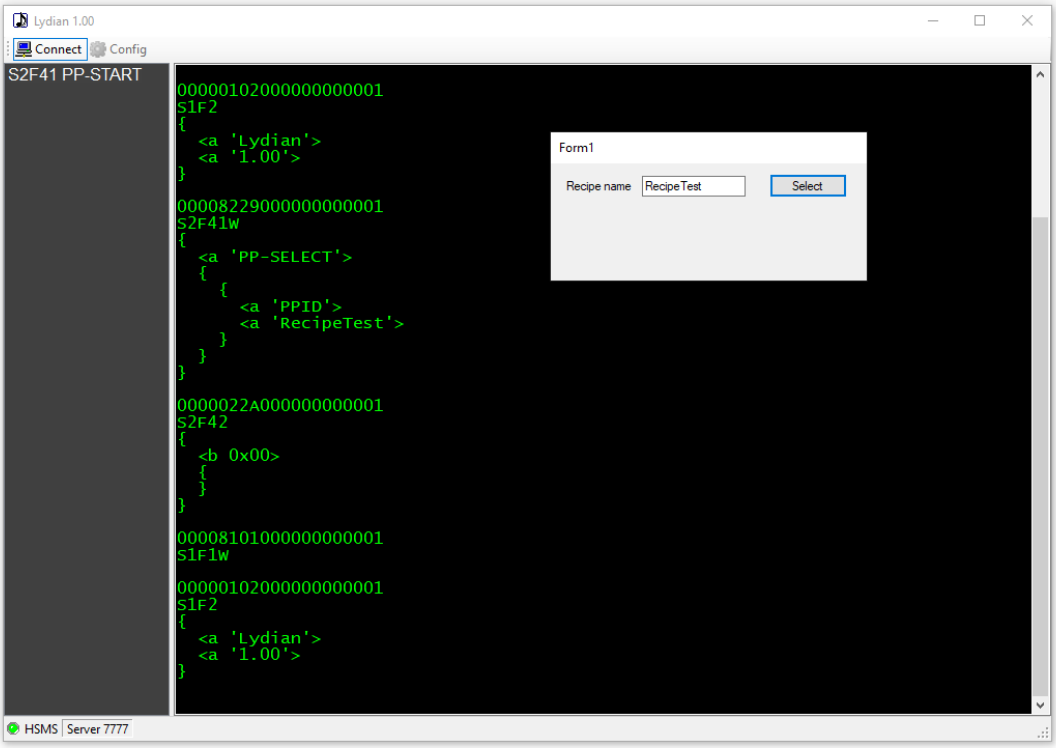
5 Double-click Select button to create the event handler function.



6 Add SML to send PP-SELECT command.

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

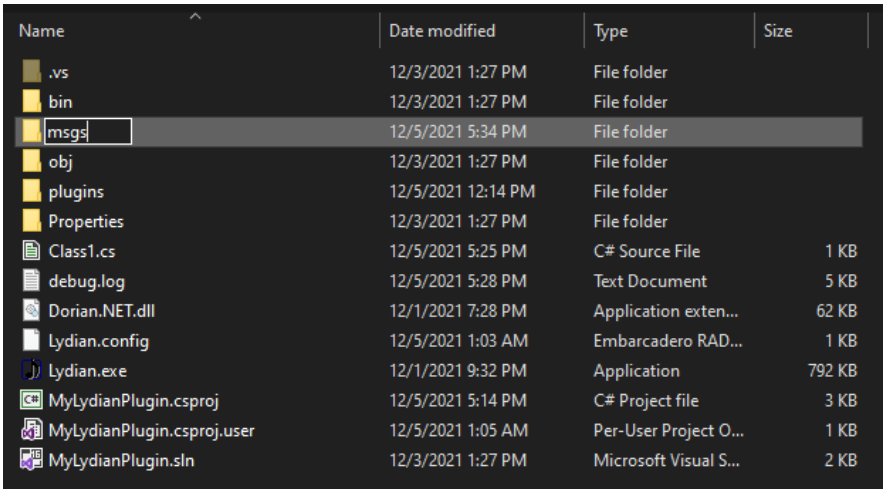
7 Build the plugin and run Lydian. When you click Select button, PP-SELECT command will be sent.



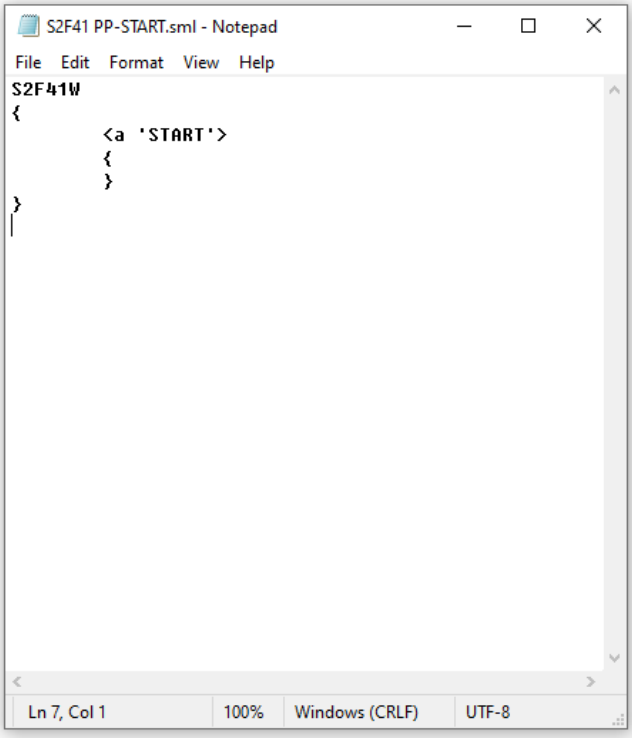
7. Message

7.1. Add Message

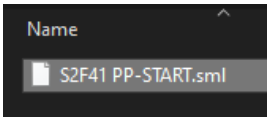
1 Create **msgs** sub-folder.



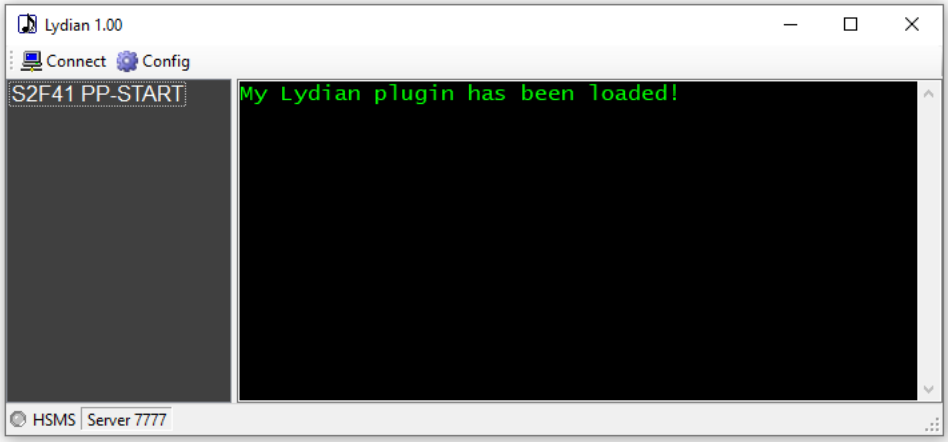
2 Use your favorite text editor (e.g. Notepad, Visual Studio, VS Code, etc) and type in a message in SML.



3 Save it to file. The file extension has to be "**.sml**".

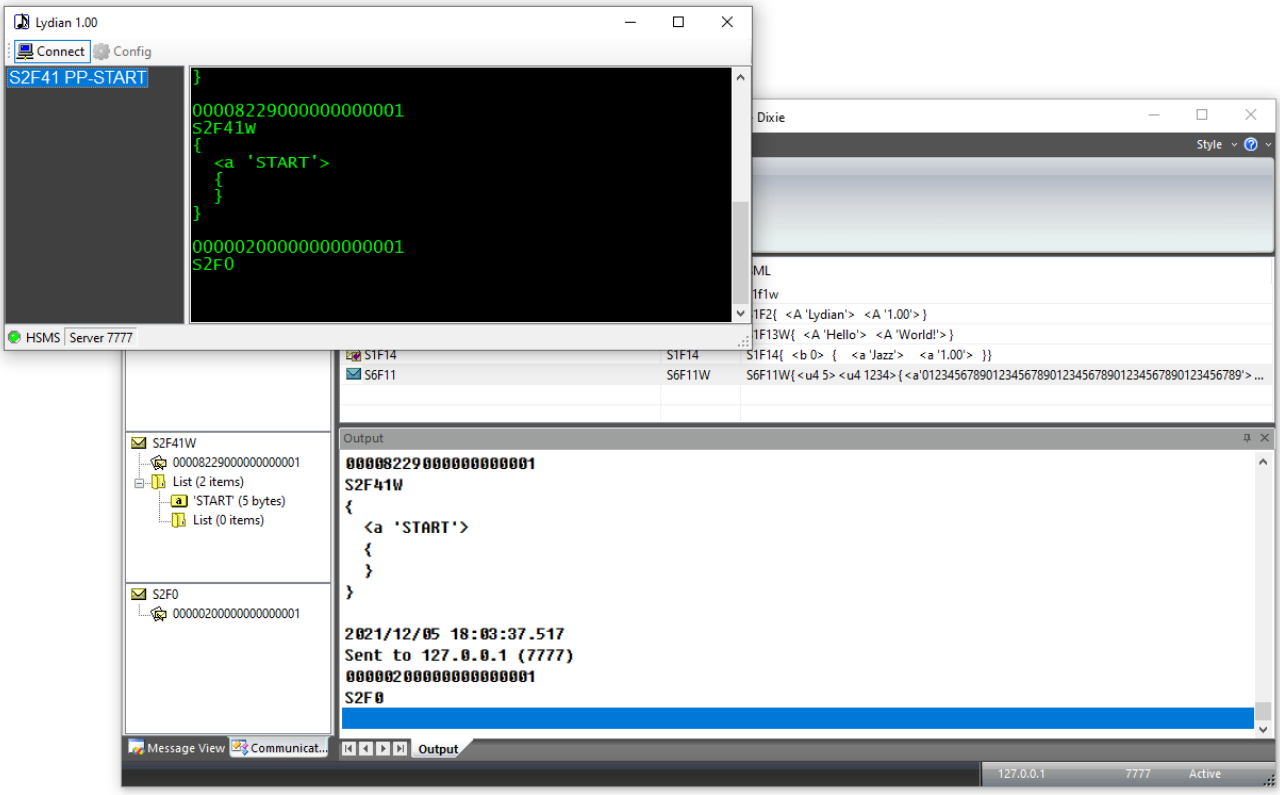


4 When you relaunch Lydian, the message you added shows up.



7.2. Send Message

1 Double-clicking the message will send out the message.



7.3. SML Reference

The grammar of the string to set Sml property is as follows;

7.3.1. Common Notice

White space (space, tab, carriage return and line feed) is treated as only a separator. It is possible use them to improve readability of source code. But it is treated as character in comment or string expression context.

From aster "*" to the end of line is treated as comment except aster in string text.

Integer consists of numeric character "0" through "9" and minus "-" flag. To write in hexadecimal expression, put "0x" in front of the expression. In this case, user can also use "a" through "f" and "A" through "F". For decimal part of the number, it is possible to omit first character "0" such like ".9" as "0.9". It also is possible to use exponential expression. There are reserved words like "true" (=1) and "false" (=0).

String is surrounded by single-quotation marks "'". It is not allowed to contain line-break and single-quotation mark itself. So if it would need to fill such kind of characters in string, use hexadecimal expression like "0x0a".

Bold letter portion in explanation means to describe character itself. These characters may be OK in either uppercase or lowercase letter. Refer to each explanation for an italic character. Moreover, the portion surrounded by brackets "[ ]" can be omitted.

7.3.2. Grammar

[sxxfyy[w]] *Body*

Item	Description
xx	Stream number. Don't insert space code between "s" and "f".
yy	Function number. Don't insert space code between "f" and "w".
w	Wait bit. Append "w" if needed.
Body	Message body.

In order to recognize stream, function, and wait-bit as 1 lump, don't put neither space nor line-break character among them. All of streams and functions can be omitted and only message body can also be described.

7.3.3. Message body

Message body is hierarchy structure.

1. List

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

Item	Description
NumOfItem	Number of list. This is only for compatibility purpose with SECSIM. SavoySecsII control would ignore this number.
Body	Message body. It is possible to insert other items here.

2. ASCII string

<a [*Strings*]>

Item	Description
Strings	ASCII literal string.

Long string can be splitted into short strings. Moreover, it is possible to use numeric character code as follows:

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

This is same as follows:

```
<a ‘ABCDEF0123456789’>
```

3. 2-byte string

2-byte string is treated as same kind of string as ASCII string. But no one saw this type in SEMI Standards specification.

<a2 [*Strings*]>

Item	Description
Strings	2-byte character string for far east complicated language. This version of Savoy control can handle only DBCS (Double Byte Character Set).

4. JIS8 string

<j [*Strings*]>

Item	Description
Strings	JIS-8 string of text for Japanese 'katakana'.

Long string can be splitted into short strings. Moreover, it is possible to use numeric character code as follows:

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

This is same as follows:

```
<j ‘ABCDEF0123456789’>
```

5. Integers

<i1 [*Numbers*]>
<i2 [*Numbers*]>
<i4 [*Numbers*]>
<i8 [*Numbers*]>
<u1 [*Numbers*]>
<u2 [*Numbers*]>
<u4 [*Numbers*]>
<u8 [*Numbers*]>

Item	Description

Numbers	Integer. It must be one of followings.

Type	Description
i1	1-byte signed integer.
i2	2-byte signed integer.
i4	4-byte signed integer.
i8	8-byte signed integer.
u1	1-byte unsigned integer.
u2	2-byte unsigned integer.
u4	4-byte unsigned integer.
u8	8-byte unsigned integer.

It is possible to enumerate multiple numbers and it means array as follows:

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

Current version of SavoySecsII cannot handle very huge number in i8 and u8.

6. Floating point number

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

Item	Description
FNumbers	Floating point number. It is one of followings.

Type	Description
f4	4-byte floating point number.
f8	8-byte floating point number.

For example,

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

7. Binary

```
<b [Numbers]>
```

Item	Description
Numbers	Numbers.

For example,

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

8. Boolean

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

<boolean [*Numbers*]>

Item	Description
Numbers	Boolean (true or false) numbers.

For example,

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

8. Lydian.IPlugin

8.1. Properties

8.1.1. lydian

Lydian object.

```
public LydianObj lydian;
```

8.2. Methods

8.2.1. Constructor

Constructor. When you create an inherited class, this has to be overridden.

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

Arguments

Name	Description
lydian	Lydian object.

8.2.2. LogMsg

Show in the message log.

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

Arguments

Name	Description
msg	SECS-II message.

If you want to write something other than SECS-II message in the message log, use log.Write() method.

8.2.3. OnReceived

This method is invoked when Lydian receives a message.

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

The inherited class has to override this method. Otherwise, it is not possible to instantiate the class object.

8.2.4. Send

Send a message set in **msgo** property. If primary message is specified, Lydian fills in the SECS-II header as a reply message.

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

Arguments

Name	Description
Primary	SECS-II primary message. If this is not null, the SECS-II header of **msgo** will be filled for the reply message based on the primary message.

Lydian sends the message and show in the message log.

9. Lydian.IPlugin.LydianObj

9.1. Properties

9.1.1. comm

Physical layer.

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

Dorian.ICommunication is a base interface of followings:

- Dorian.Hsms
- Dorian.Forms.Hsms
- Dorian.SecsII
- Dorian.Forms.SecsII

Lydian uses Dorian.Forms.Hsms and Dorian.Forms.SecsII.

9.1.2. DeviceID

Device ID of SECS-II header.

```
public ushort DeviceID;
```

9.1.3. handle

Lydian form handle.

```
public IWin32Window handle;
```

9.1.4. Host

Host or equipment.

```
public bool Host;
```

9.1.5. Hsms

HSMS or SECS-I.

```
public bool Hsms;
```

9.1.6. log

Communication log.

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

Dorian.ILog is a base interface of followings:

- Dorian.Log
- Dorian.Forms.Log
- Dorian.Forms.LogView

Lydian uses Dorian.Forms.LogView.

9.1.7. **msgi**

Incoming message.

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

Dorian.ISecsII is a base interface of followings:

- Dorian.SecsII
- Dorian.Forms.SecsII

Lydian uses Dorian.Forms.SecsII.

9.1.8. **msgo**

Outgoing message.

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

9.1.9. **SystemBytes**

System bytes of SECS-II.

```
public uint SystemBytes;
```

This is an unique identifier of the SECS-II message and has to be incremented or different everytime sending a message.