

RYSKAMP LEARNING MACHINE

Developer's Guide

Chapter 1: Download and Installation

1.1. Checking Requirements

We have made an easy way for you to check the compatibility of your machine, which is located at the downloaded directory:

```
.\ExampleApps\UtilityTools\RequirementsTool\bin\debug\RequirementsTool.exe
```

It should check if you want to use the python code or not and immediate check if there are any compatibility issues and advise what to do.

- a. **Visual Studio 2015**
 - We have not tested it on the 2017 Visual Studio, please let us know if there are errors.
- b. [SQL Server 2014 and up](#)
- c. [.Net Framework 4.6.2](#)
- d. If are using Python Code (OPTIONAL)
 - [Python 3.5](#)
 - [Python Tools 2.2.6 for Visual Studio 2015](#)
 - Python .NET ([explained in the guide](#))
- e. Recommended Machine Requirements
 - CPU: Quad Core
 - RAM: 8 GB
 - OS: Windows 8 or 10
- f. Minimum Machine Requirements
 - CPU: Single Core
 - RAM: 2 GB
 - OS: Windows 8 or 10

```
System Requirements Check:

Are you using Python? y/n
y

Python... OK
PythonNET...OK


.NET Framework...OK
MS SQL Server...OK
Visual Studio...OK

Windows 8 or 10 versions... OK
RAM >= 2Gb... OK
-
```

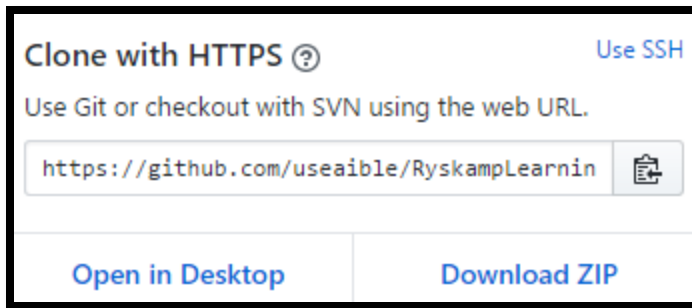
1.2. Downloading the Source Code

To get the source code, please go to our **Github Repository** at <https://github.com/useaible/ryskampllearningmachine>

Once you see the Repository, click Clone or Download.

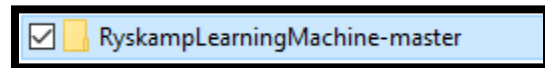


Then click **Download Zip**.



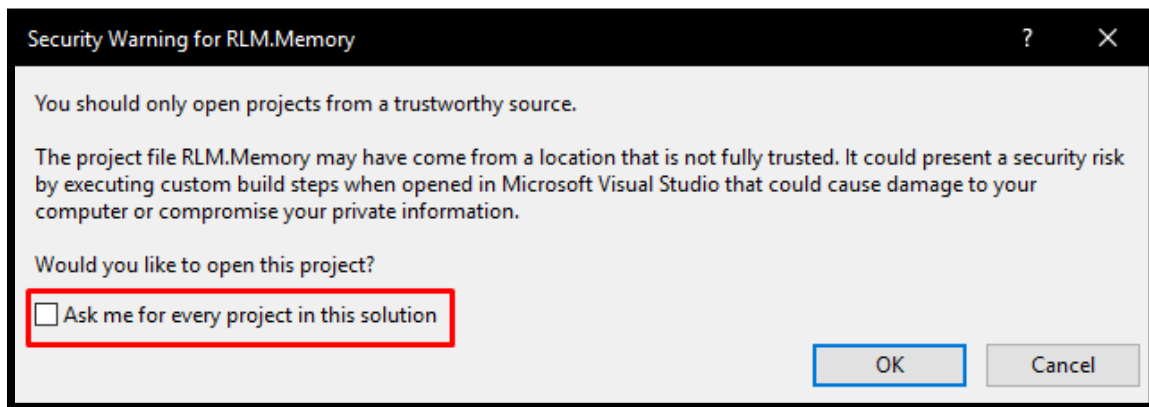
1.3. First Opening the Solution

Once the download is finished, Extract the zip file and you should now have a new folder named **RyskampLearningMachine-master**

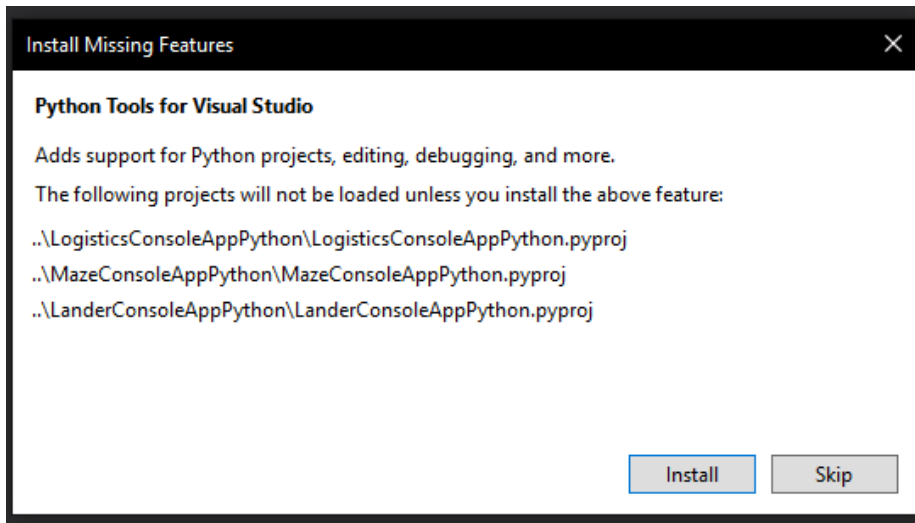


This should contain some files, one of which is **RLM.sln**, you will need to open it via Visual Studio.

You may run into this error below. You may just simply uncheck the **Ask me for every project in this solution** then click on OK to proceed.



Also, since the Solution includes Python code, in the event that you have not installed Python Tools 2.2.6 for Visual Studio 2015, you will be prompted to install Python Tools for Visual Studio. **THIS IS OPTIONAL BUT IF YOU WANT TO USE THE PYTHON GAMES WE RECOMMEND INSTALLING IT EXTERNALLY THROUGH DOWNLOADABLE INSTALLER [HERE](#).**

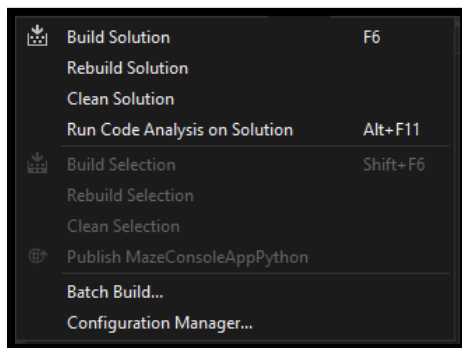


1.4. Building the Solution

You will now want to build the code. This will download any missing packages and make the code run for the included applications. To build the code in Visual Studio, click on Build.



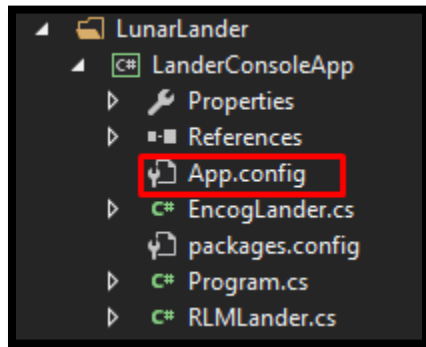
Then click on Build Solution or you could simply press F6. This might take a few seconds to a minute.



1.4.1. Connection Strings

Once the solution has been rebuilt, you may want to verify your connection strings. By default, it uses the SQL Server installed locally on your computer. You can configure this via the App.config for each of the Sample Apps.

Note that this has to be done for each of the app that you want to run.



You will need to configure the line below to your needs.

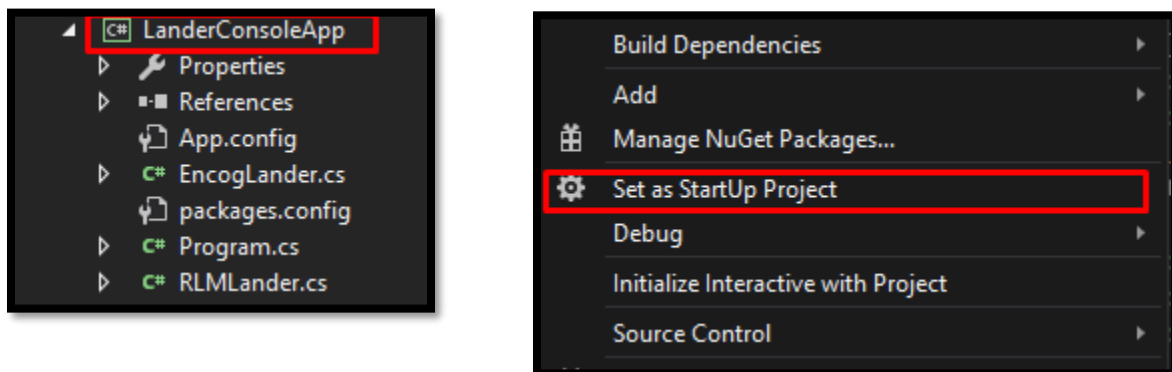
```
<appSettings>
  <add key="RLMConnStr" value="Server=.;Database={{dbName}};Integrated Security=True;" />
  ...
</appSettings>
```

Another item for the App Settings are the Backup and Data location. By default, folders shown below are created. You may configure this as you see fit.

```
<add key="RLMBackupLocation" value="C:\RLM\Backup" />
<add key="RLMDataLocation" value="C:\RLM\Data" />
```

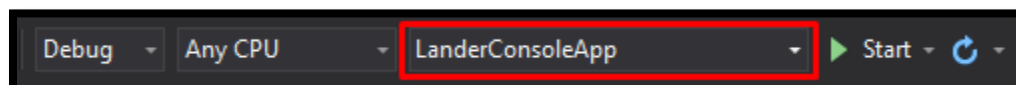
1.5. First Running a Sample App

We should now be able to run the code. You can choose what your Startup Project is and Run that. On your Solution explorer, we can setup the LanderConsoleApp as the Startup Project.



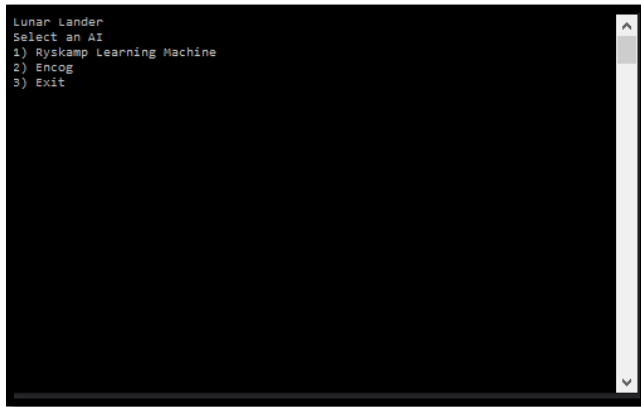
Right click on it and Select Set as Startup Project.

It should then show up as the Startup project located at the top of the page



Just click on Start and it should run.

Here we run the Lander Console App (Lunar Lander). For any errors, please refer to common errors section.

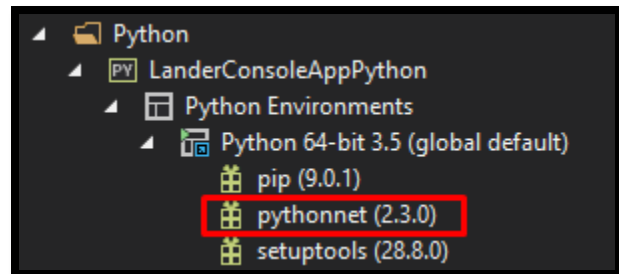


1.6. Running the Python Sample Apps

1.6.1. Installing Python.Net

If you don't have it already, you might be missing python net.

But with the Python Environment already installed, it should be easy. Just get the Scripts folder directory of your Python Environment and run from Command Prompt with Administrative Privileges.



The command is `pip install pythonnet`.

```
\AppData\Local\Programs\Python\Python35-32\Scripts>pip install pythonnet
```

It should go through the install process.

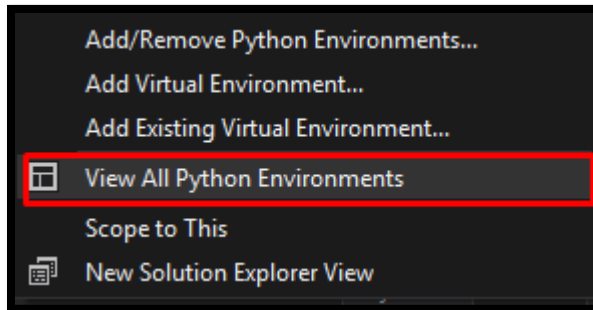
```
Collecting pythonnet
  Downloading pythonnet-2.3.0.tar.gz (1.5MB)
    100% |#####| 1.5MB 172kB/s
Installing collected packages: pythonnet
  Running setup.py install for pythonnet
Successfully installed pythonnet-2.3.0
You are using pip version 7.1.2, however version 9.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

For errors, check section 1.7. [Common Errors](#).

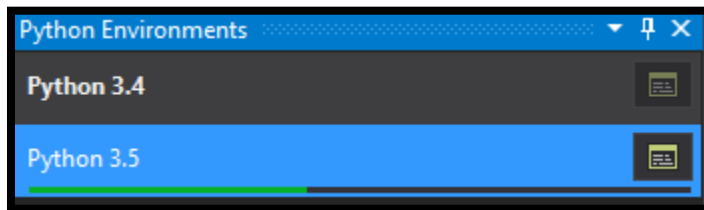
1.6.2. Python Environments

If there are no available options under Python Environments and it's already installed, you might need to refresh it.

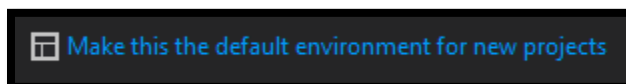
Right click the Python Environments under your selected Sample Python App. It should show up an option to view all python environments.



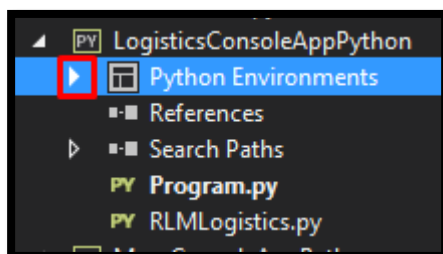
At the moment, we highly recommend Python 3.5. Just refresh it and it should load. If it's the only Python Environment you have, it should automatically populate.



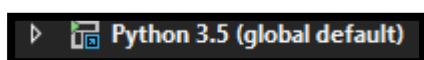
Once that is done loading, make it your default by clicking the link below it as shown below



To confirm that everything has been done correctly, go back to the solution explorer, and you should now see a drilldown for the Python Environments



Once clicked, it should show your selected environment.



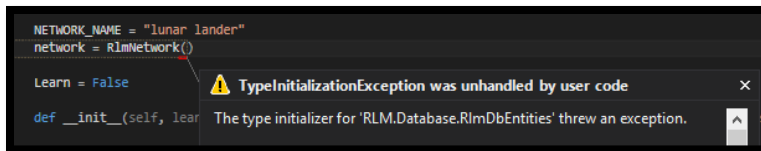
1.7. Common ERRORS

1.7.1. Missing Database

C#

```
ERROR: The type initializer for 'RLM.Database.RlmDbEntities' threw an exception.
```

Python

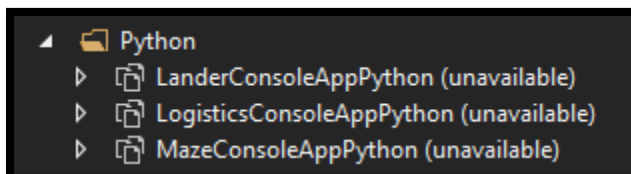


Solution:

You're SQL might not be turned on or existing. You can check this through services.msc

SQL Full-text Filter Daemon Launcher (MSSQLSERVER)	Running	Automatic	NT Service...
SQL Server (MSSQLSERVER)	Running	Automatic	NT Service...
SQL Server Agent (MSSQLSERVER)		Automatic	NT Service...

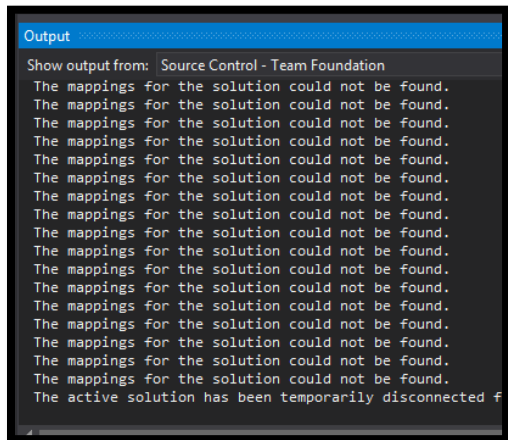
1.7.2. Python Tools Not Installed



Solution:

You can install Python Tools which can be downloaded [HERE](#). Once installed, just relaunch Microsoft Visual Studio.

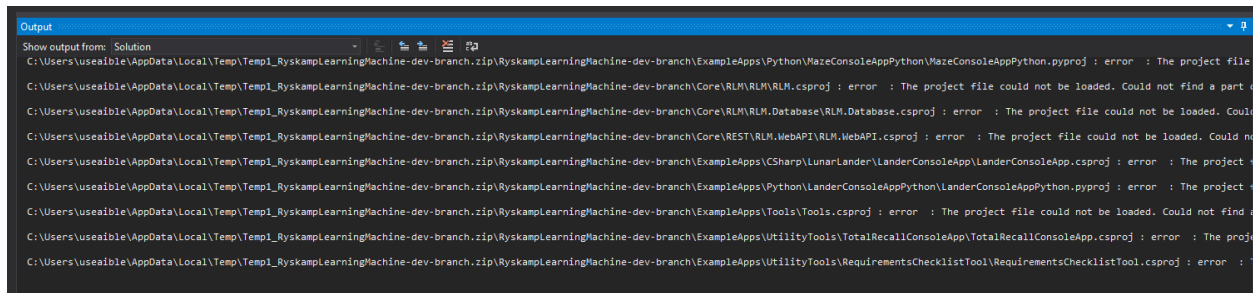
1.7.3. The Mappings for the Solution could not be found



Solution:

If you're getting his type of error on your output after first opening it, just disregard it for now.

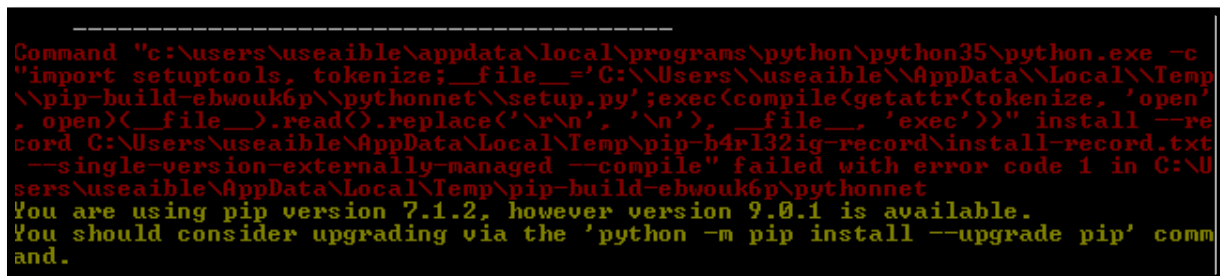
1.7.4. Project File cannot be loaded



Solution:

You will need to extract the solution first from the zip file to be able to open the project without problems.

1.7.5. Error while installing Python.Net through Command Prompt



Solution:

This is an error easily remedied by updating your Python PIP. To do so, type the command below on the Scripts directory

pip install --upgrade pip

Note that there are two dashes (not -, but --)

It should update, then after which you can proceed to installing Python.Net with the command

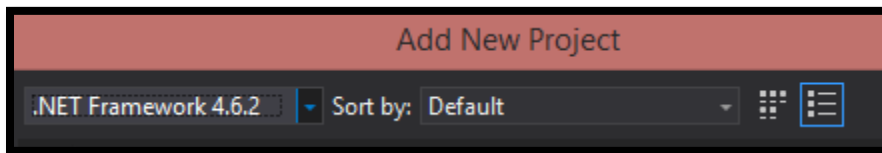
pip install pythonnet

Chapter 2: The RLM Setup

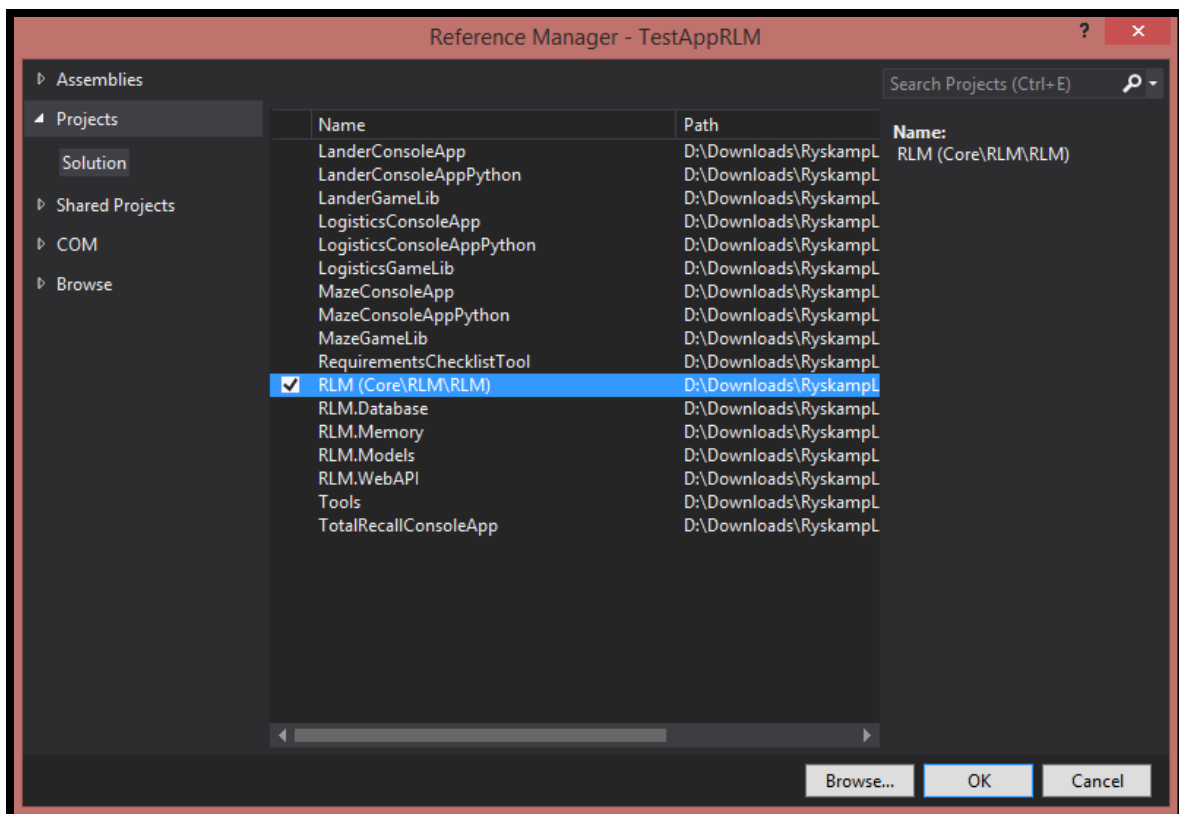
2.1. Setting Up References

We want you to also set up your own game to try with our RLM. So for us to do that, we're going to walk you through setting up your network and training the RLM to fit your needs. You'll be glad to know that it is very easy.

If you want to create a sample project, don't make it under the core folder. You will also need to set it to .Net Framework 4.6.2



Once you've created your project, add the references. The most important one is the RLM Core



2.2. Setup the Database

Now that you're done setting up the references we will need to start coding.

The first item that needs to be setup is the network object, in which the `rlm_network` class is used. This class controls the network creation as well as the RLM Objects training state.

If setting up with no database yet, you could use a default constructor which will automatically create a "RyskampNeuralNetworks" database.

```
var rlmNet = new RlmNetwork();
```

Above shows us setting up a new network object named `rlmNet`. This will then create an "RyskampNeuralNetworks" database which will be tied to the created network.

If you already have a database setup, you can instead use the constructor with the database name string parameter.

```
var rlmNet = new RlmNetwork("MyDatabaseName");
```

This will create a new a new network object and with it, the new database with the name **MyDatabaseName**.

You will also need to subscribe to two events which triggers if the data is fully written to the database.

```
network.DataPersistenceComplete += Network_DataPersistenceComplete;
network.DataPersistenceProgress += Network_DataPersistenceProgress;
```

2.3. Setting up the Inputs and Outputs

The second step is to identify the Inputs and Outputs and set them up in a way that the RLM is able to identify and use them for training.

```
public RlmIO(String name, String dotnettype, double min, double max,
Enums.RlmInputType type, long id = 0)
```

You will have to make a list of the `RlmIO` object as the method only accepts a list.

```
var inputs = new List<RlmIO>();
inputs.Add(new RlmIO("RlmInputOne", typeof(bool).ToString(), 0, 1,
RlmInputType.Distinct));
inputs.Add(new RlmIO("RlmInputTwo", typeof(bool).ToString(), 0, 1,
RlmInputType.Distinct));
```

Above you can see that we have made a new list of inputs. First we create the list and then we fill it up with `RlmIO` objects, which we defined above as `RLMInputOne` and `RLMInputTwo`, both are set to boolean with `.Net` type and are `Distinct`. We have not set IDs, these are optional.

Outputs are made in the same way yet a little different. For it, we will be using a different constructor for the RlmIO object.

```
public RlmIO(String name, String dotnettype, double min, double max,
long id = 0)
```

This will not have an RLM input type being that it's not an input.

```
var outputs = new List<RlmIO>();
outputs.Add(new RlmIO("XOROutput", typeof(bool).ToString(), 0, 1));
```

Similar to how we made the inputs list, we made the outputs list and added a new output RLMOutputOne. Note that there is no RLM Input type parameter.

2.4. Loading the Network

Loading a Network means to save in memory all the saved settings for that network. To load the network, we need to check first if a network is already existing and that if it is, we should load that instead.

```
rlmNet.LoadNetwork("MyNetworkName");
```

If there's no network of the specified name, then we should create a new Network. However, the LoadNetwork method does not make a new network but only returns a boolean type, so we will need to use the NewNetwork method.

```
rlmNet.NewNetwork("MyNetworkName", inputs, outputs);
```

The NewNetwork method sets up a network together with its inputs and outputs. As we've discussed in the previous section, the lists of inputs and outputs are going to be used as parameters. We will then need to combine both in an IF statement.

```
if (!rlmNet.LoadNetwork("MyNetworkName"))
    rlmNet.NewNetwork("MyNetworkName", inputs, outputs);
```

Here we check if MyNetworkName is existing and if it does, we load it, however, if it is not existing, then we create a new one.

2.5. Additional Settings

Part of the network setup are settings that help how much efficiently the engine trains. This would be the percentage of randomness, the number of sessions, and for linear problems, the linear bracket.

The NumSessions is an int type that determines how many times the RLM trains. The more sessions, the more opportunities for the engine to learn and apply its learning.

```
rlmNet.NumSessions = 50;
```

Shown above is how to set up the NumSessions to 50 sessions.

The Randomness Property, defined with its min and max, dictates how much randomness is applied on the training of the engine. As it moves forward to more sessions, the randomness depreciates which allows the AI to apply its learning more as opposed to learning something new until it reaches its set EndRandomness.

```
rlmNet.StartRandomness = 100;  
rlmNet.EndRandomness = 0;
```

Here we see the engine is set to start at 100% randomness, allowing it to have no limitations in trying different things, of which will depreciate as it moves further through its sessions until it reaches the last session which then its Randomness will be 0, making it only apply its learning.

Another setting is the Linear Bracket used for Linear Inputs. A good indication on when to use this is when an input's value is large and granular enough that you may consider close neighboring values to be the same depending on the linear bracket.

```
rlmNet.MaxLinearBracket = 15;  
rlmNet.MinLinearBracket = 3;
```

Here we set the Min and Max to 3 and 15, respectively. In our testing, this has worked with the best results.

Another example would be, the values 7 and 8 will be processed by the RLM similarly if those two numbers are within the linear bracket at that time. But if this input was set as a Distinct Input then 7 and 8 would be two different values and processed differently by the RLM. The linear bracket (Max - Min) decreases after each session and is applied at each cycle when getting the best solution.

There is no perfect number setting for the Linear Bracket, it's more of a trial and error and check what setting allows the engine to learn more efficiently.

Chapter 3: The RLM Training

3.1. The XOR App

We believe the best way to explain the concept is through example. So in this chapter, in order for us to explain training, we will be making and training the RLM to analyze and learn the XOR App.

The XOR, or Exclusive OR, is a truth table that indicates TRUE whenever the items being compared are different. Example, 1 XOR 2, would be TRUE. However, 3 XOR 3 would be FALSE.

With that known, we are going to have the AI learn and predict XOR. Note that the XOR App is also included with the downloadable source code.

3.2. The Training Setup

We will need to setup the references and connection strings to start. Create a project, we recommend a console application, name it accordingly and we can start adding the references.

Right-click references on your solution explorer for your new solution, then click Add References. Then select the RLM Core and the other libraries.



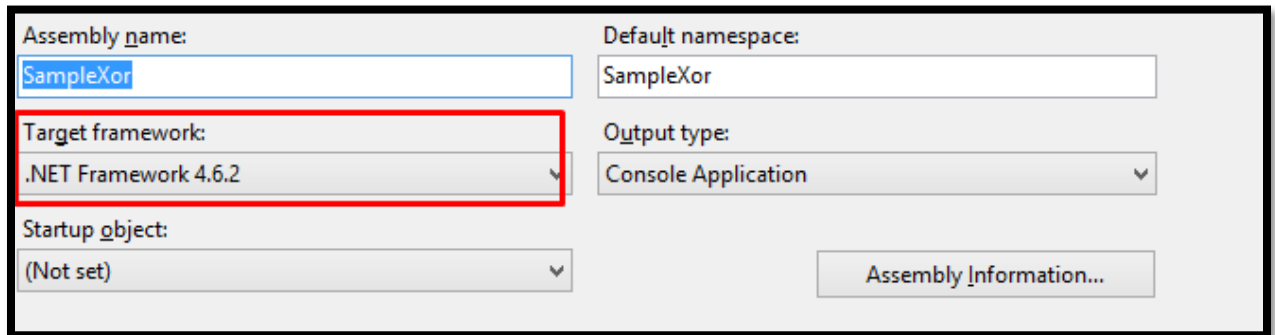
Once that has been added, we need to configure the connection strings for your local SQL Server. Open App.config by double-clicking on it. Add the App settings below:

```
<appSettings>
  <add key="RLMConnStr" value="Server=.;Database={{dbName}}"/>
</appSettings>
```

If you don't have a local server, you can change the string of the value variable and change it to your liking. The server can be changed. If you have a username and password for that SQL Server, add the keywords after you add a semi-colon (;) below:

```
;User=SampleUsername;Password=SamplePassword
```

As mentioned on Chapter 2, we will need to use .Net Framework 4.6.2 so let's make sure that your new solution is targeting that. Right click on your solution name and go to properties. On the Application tab, you should see Target Framework. Make sure that it is 4.6.2.



Then finally, the namespace need to be defined.

```
using RLM.Models;
using RLM.Enums;
using RLM;
```

Let's start coding!

Our first approach would be identifying the inputs as well as the outputs. We have 2 inputs, this will be the items to compare, and 1 output, which will be the result.

We will also need the ideal data, of which our training will be based upon. For this, we will use the XOR table which contains the possibilities of inputs and with it, its correct output. Let's declare this outside of our main program.

```
static IEnumerable<dynamic> xorTable = new List<dynamic>
{
    new { Input1 = "True", Input2 = "True", Output = "False" },
    new { Input1 = "True", Input2 = "False", Output = "True" },
    new { Input1 = "False", Input2 = "True", Output = "True" },
    new { Input1 = "False", Input2 = "False", Output = "False" },
};
```

Let's start coding the program.

```
var rlmNet = new RlmNetwork($"RLM_XOR_SAMPLE_{Guid.NewGuid().ToString("N")}");
```

We have appended a GUID here so that we have a unique RLM Network everytime we run this example, to have a fresh start. You can remove this or simply change the name to something static to use the same network all the time.

```
rlmNet.DataPersistenceComplete += RlmNet_DataPersistenceComplete;
rlmNet.DataPersistenceProgress += RlmNet_DataPersistenceProgress;
```

Here are some events that we can subscribe to. The DataPersistenceComplete notifies as if the DB

Write is done. DataPersistenceProgress shows us what the progress is and of what total. Below is a sample method that runs which are subscribed to the events. These will display on the console once it starts.

```
private static void RlmNet_DataPersistenceComplete()
{
    Console.WriteLine("Done");
}

private static void RlmNet_DataPersistenceProgress(long processed, long total)
{
    Console.WriteLine("{0} of {1}", processed.ToString(), total.ToString());
}
```

We will now be loading the network. This is not a database, but a table in your defined database.

```
if (!rlmNet.LoadNetwork("XOR_SAMPLE"))
{
    // here you declare our inputs
    var ins = new List<RlmIO>();
    ins.Add(new RlmIO("XORInput1", typeof(bool).ToString(), 0, 1,
        RlmInputType.Distinct));
    ins.Add(new RlmIO("XORInput2", typeof(bool).ToString(), 0, 1,
        RlmInputType.Distinct));

    // and here, declare our outputs
    var outs = new List<RlmIO>();
    outs.Add(new RlmIO("XOROutput", typeof(bool).ToString(), 0, 1));

    // creates a new network
    rlmNet.NewNetwork("XOR_SAMPLE", ins, outs);
}
```

Here we have declared our inputs and outputs, all of the type boolean, as well as creating the network. This code is enclosed in an if statement that checks if there's already a network created for the App.

Now, we will need to set the network settings.

```
rlmNet.NumSessions = 50;
rlmNet.StartRandomness = 50;
rlmNet.EndRandomness = 0;
```

We're just going to start with 50 sessions, which is not too many yet good enough to squeeze some learning in. The StartRandomness is the percentage of randomness applied to the engine, to try new values or not. This will get lower as the sessions move up until it reaches the EndRandomness set.

3.3. Starting the Session

```
Console.WriteLine("Training Session started");
```

To start the session, we will be using the method `SessionStart()` to set the network's status to Started and return the Session ID, which we will then store. We will be enclosing this on a for loop that will reiterate until all the sessions are done.

```
for (int i = 0; i < sessions; i++)
{
    long sessionId = rlmNet.SessionStart();
}
```

With the network now started, we will be coding a for loop to iterate through all the sessions while learning.

3.4. Processing Your Data

Inside the for loop, we will be reiterating our engine to each of the values on the our XOR table and allow it to answer and we score.

```
double sumOfCycleScores = 0;

foreach (var xor in xorTable)
{
    //Populate input values
    var invs = new List<RlmIOWithValue>();

    // get value for Input1 and associate it with the Input instance
    string input1Value = xor.Input1;
    invs.Add(new RlmIOWithValue(rlmNet.Inputs.Where(item => item.Name ==
        "XORInput1").First(), input1Value));

    // get value for Input2 and associate it with the Input instance
    string input2Value = xor.Input2;
    invs.Add(new RlmIOWithValue(rlmNet.Inputs.Where(item => item.Name ==
        "XORInput2").First(), input2Value));

    // Once you have generated your input values, we will need to process them. We
    // will be using RlmCycle class which handles processing of training data. thin
    // this class is the RunCycle method which has return type of
    // RlmCyclecompleteArgs, which is an object type that stores cycle outputs of
    // the rlm network.
    var Cycle = new RlmCycle();
    RlmCyclecompleteArgs result = Cycle.RunCycle(rlmNet, sessionId, invs, true);

    // scores the RLM on how well it did for this cycle
    // each cycle with a correct output is rewarded a 100 score, 0 otherwise
    // we will define this method below
    double score = ScoreCycle(result, xor);

    sumOfCycleScores += score;
}
```

```

    // sets the score on the Database
    rlmNet.ScoreCycle(result.CycleOutput.CycleID, score);
}

Console.WriteLine($"Session #{i} - score: {sumOfCycleScores}");

```

Below is a user made method and will depend on how the user decides on scoring the RLM's decision.

```

static double ScoreCycle(RlmCyclecompleteArgs cycleResult, dynamic xor, bool
showOutput = false)
{
    double score = 0;
    string output = null;

    // get output
    output = cycleResult.CycleOutput.Outputs.First().Value;

    // check if RLM output is correct based on our xor reference
    if (xor.Answer == output)
    {
        score = 100;
    }

    if (showOutput)
    {
        Console.WriteLine($"{ xor.Input1 } and { xor.Input2 } is { output }");
    }

    return score;
}

```

We simply check the output is correct based on the XOR table we setup, and if it is, we give the score 100, else, it will default to 0.

We will then need to end the Session and add the score from sumOfCycleScores where we summed up the Cycle Scores.

```

    rlmNet.SessionEnd(sumOfCycleScores);
} // this is also where the loop of the sessions end

```

3.5. Prediction

After processing the data, we will now check if our engine has learned from the training session. Note that the only difference with Predict from training is that we put 'false' to the learn argument on the Cycle.RunCycle() method and of course, the inputs which we used our xor table to check if the RLM has learned as expected. This should be outside the training sessions loop.

```

long SessionID = rlmNet.SessionStart();
sumOfCycleScores = 0;

foreach (var xor in xorTable)
{
    var invs = new List<RlmIOWithValue>();
    invs.Add(new RlmIOWithValue(rlmNet.Inputs.First(a => a.Name == "XORInput1"),
    xor.Input1));
    invs.Add(new RlmIOWithValue(rlmNet.Inputs.First(a => a.Name == "XORInput2"),
    xor.Input2));

    RlmCycle Cycle = new RlmCycle();
    RlmCyclecompleteArgs result = Cycle.RunCycle(rlmNet, SessionID, invs, false);

    double score = ScoreCycle(result, xor, true);

    sumOfCycleScores += score;

    // sets the score
    rlmNet.ScoreCycle(result.CycleOutput.CycleID, score);
}
rlmNet.SessionEnd(sumOfCycleScores);

```

Then we will need to call the TrainingDone() method to notify all events that we are done, and the Data Persistence is notified

```

rlmNet.TrainingDone();
Console.ReadLine();

```

And finally, we have the output:

```

Training Session started
Session #0 - score: 300
Session #1 - score: 300
Session #2 - score: 300
Session #3 - score: 400
Session #4 - score: 400
Session #5 - score: 200
Session #6 - score: 100
Session #7 - score: 400
Session #8 - score: 200
Session #9 - score: 300
Session #10 - score: 200
Session #11 - score: 300
Session #12 - score: 300
Session #13 - score: 400
Session #14 - score: 300
Session #15 - score: 400
Session #16 - score: 300
Session #17 - score: 400
Session #18 - score: 300
Session #19 - score: 400
Session #20 - score: 400
Session #21 - score: 400
Session #22 - score: 400
Session #23 - score: 200
Session #24 - score: 300
Session #25 - score: 400
Session #26 - score: 300
Session #27 - score: 400
Session #28 - score: 400
Session #29 - score: 400
Session #30 - score: 400
Session #31 - score: 300
Session #32 - score: 300
Session #33 - score: 400
Session #34 - score: 300
Session #35 - score: 300
Session #36 - score: 400
Session #37 - score: 400
Session #38 - score: 400
Session #39 - score: 300
Session #40 - score: 400
Session #41 - score: 400
Session #42 - score: 400
Session #43 - score: 300
Session #44 - score: 400
Session #45 - score: 300
Session #46 - score: 400
Session #47 - score: 400
Session #48 - score: 400
Session #49 - score: 400
Training Session ended
Predict:
True and True is False
True and False is True
False and True is True
False and False is False

```