

# 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
-
```

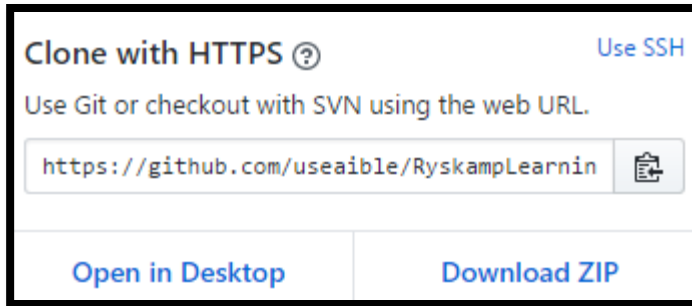
## 1.2. Downloading the Source Code

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.

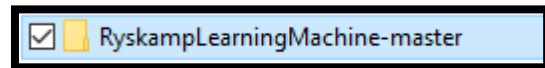


Then click **Download Zip**.

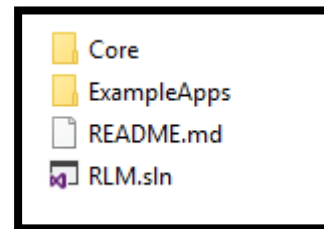


### 1.3. First Opening the Solution

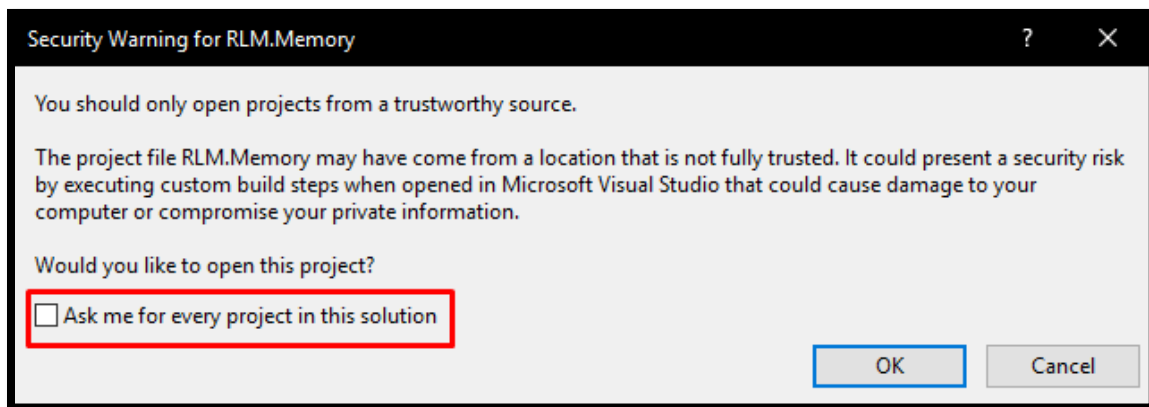
Once the download is finished, Extract the zip file and you should now have a new folder named **RyskampLearningMachine-master**



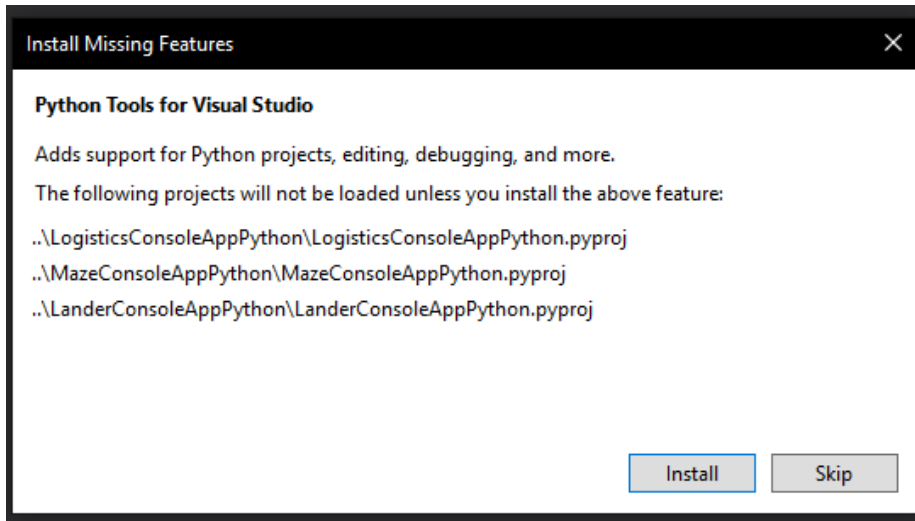
This should contain 2 folders and 2 files, one of which is **RLM.sln**, you will need to open it via Visual Studio.



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](#).**

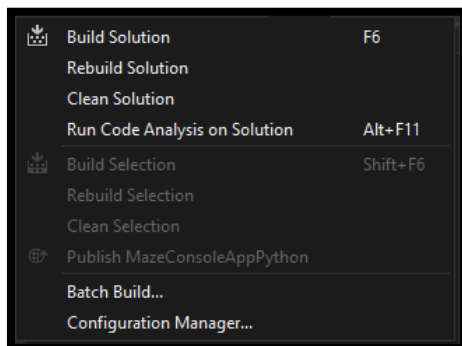


## 1.4. Building the Solution

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



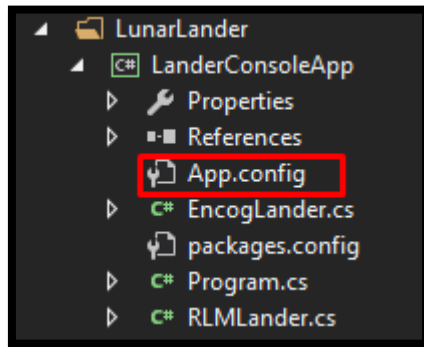
Then click on Build Solution or you could simply press F6. This might take a few seconds to a minute.



### 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.

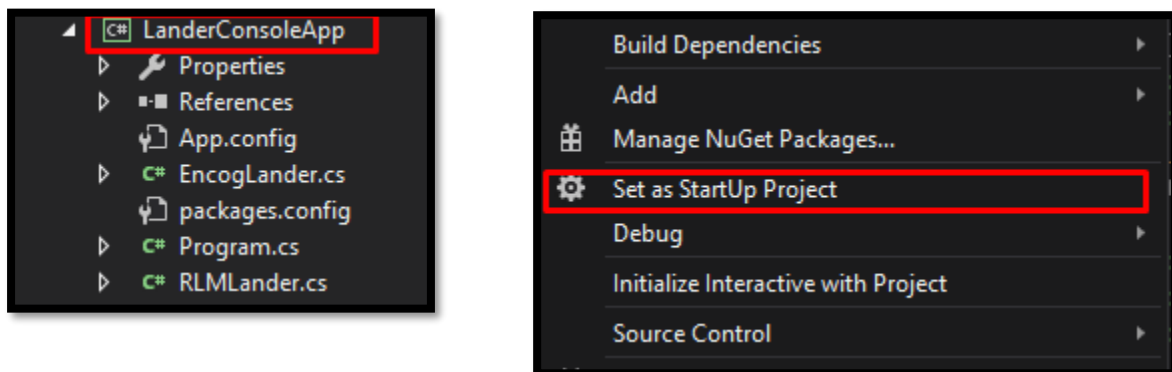
```
<appSettings>
  <add key="RLMConnStr" value="Server=.;Database={{dbName}};Integrated Security=True;" />
  ...
</appSettings>
```

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" />
```

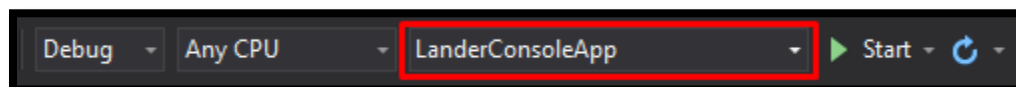
## 1.5. First Running a Sample App

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.



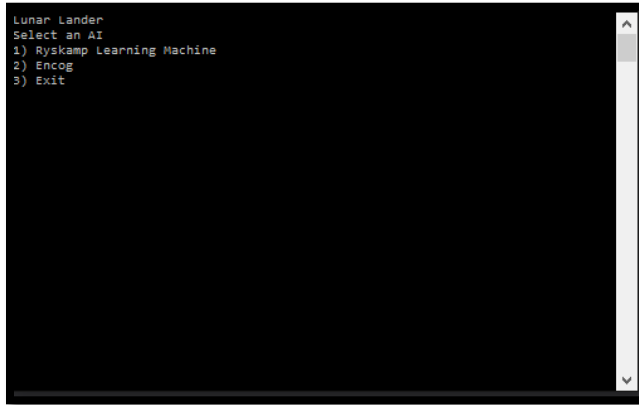
Right click on it and Select Set as Startup Project.

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.

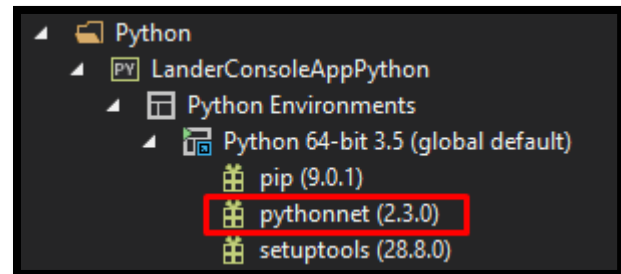


## 1.6. Running the Python Sample Apps

### 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 python.net`.

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

It should go through the install process.

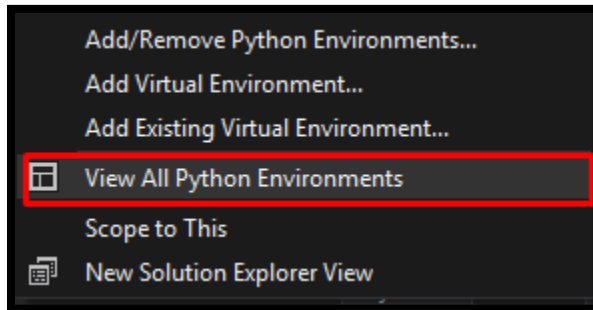
```
Collecting pythonnet
  Downloading pythonnet-2.3.0.tar.gz (1.5MB)
    100% |#####| 1.5MB 172kB/s
Installing collected packages: pythonnet
  Running setup.py install for pythonnet
Successfully installed pythonnet-2.3.0
You are using pip version 7.1.2, however version 9.0.1 is available.
You should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

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

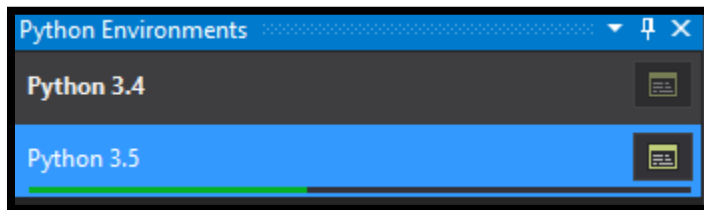
### 1.6.2. Python Environments

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

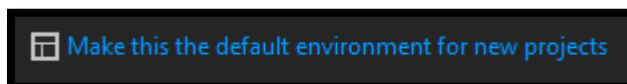
Right click the Python Environments under your selected Sample Python App. It should show up an option to view all python environments.



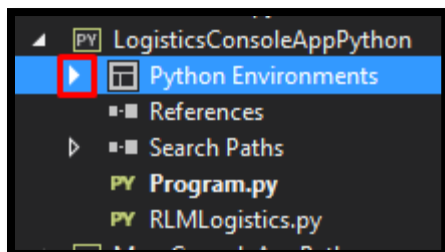
At the moment, we highly recommend Python 3.5. Just refresh it and it should load. If it's the only Python Environment you have, it should automatically populate.



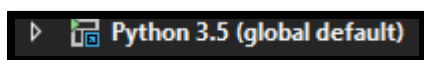
Once that is done loading, make it your default by clicking the link below it as shown below



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.



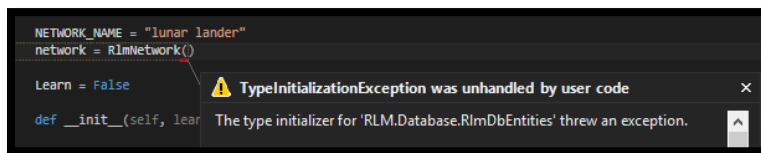
## 1.7. Common ERRORS

### 1.7.1. Missing Database

#### C#

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

#### Python

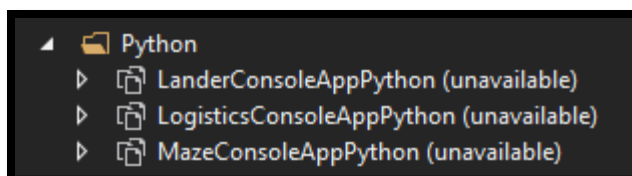


#### Solution:

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

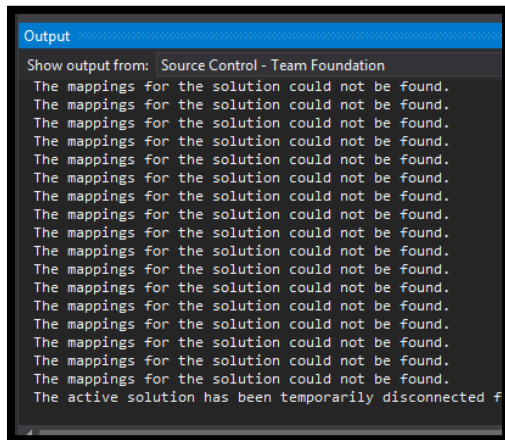


#### Solution:

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



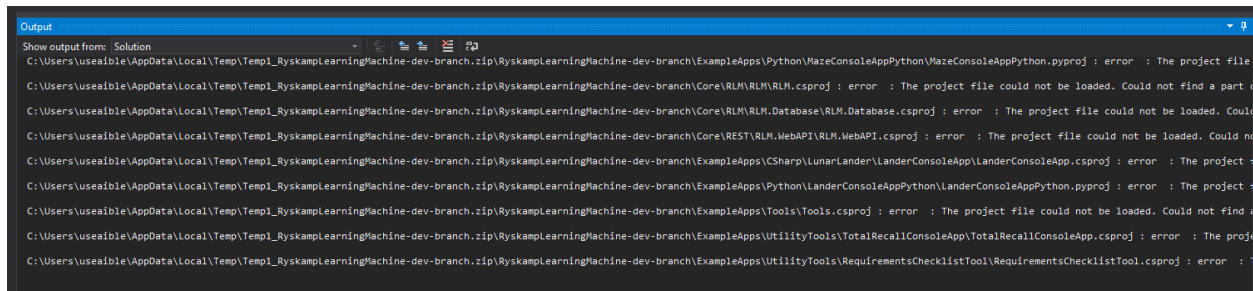
### 1.7.3. The Mappings for the Solution could not be found



#### Solution:

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

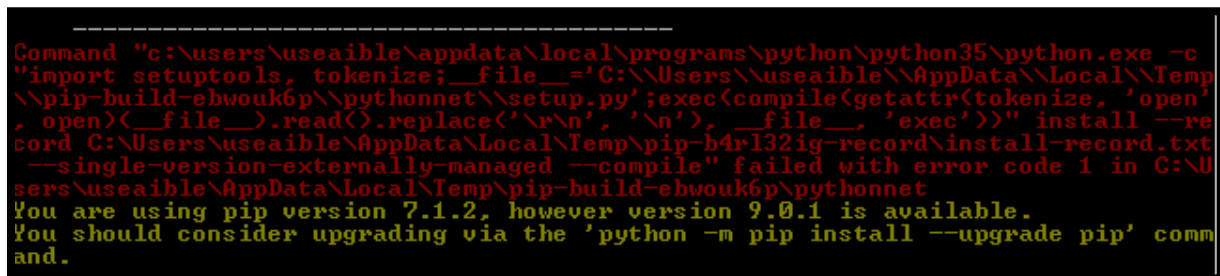
### 1.7.4. Project File cannot be loaded



#### 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

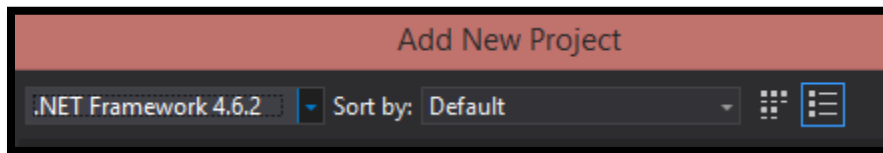
**pip install pythonnet**

# Chapter 2: The RLM Setup

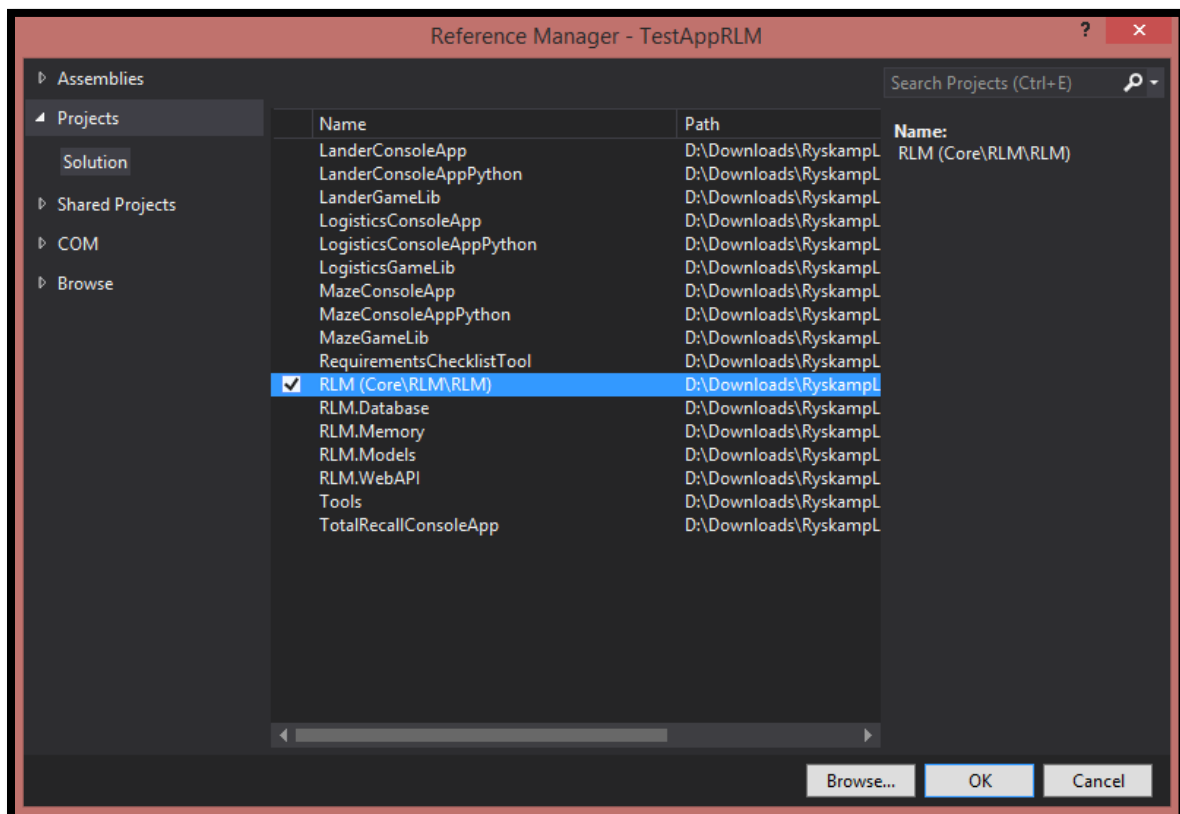
## 2.1. Setting Up References

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 for 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`.

## 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 `rlm_io` object as the method only accepts a list.

```
var inputs = new List<RlmIO>();
inputs.Add(new RlmIO("XORInput1", typeof(bool).ToString(), 0, 1,
    RlmInputType.Distinct));
inputs.Add(new RlmIO("XORInput2", 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 `rlm_io` objects, which we defined above as `RLMInputOne` and `RLMInputTwo`, both are set to boolean .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 `rlm_io` 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 save 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 on 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. Note that each session

```
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. In summary, this allows you to work with Linear Values or multiple inputs.

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

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

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 the training will be based upon. For this, we will use the XOR table which contains the possibilities of inputs with it, its correct output.

```
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" },
};
```

For a XOR Comparison, we will need something simple, which the boolean type is perfect. Once that's identified, let's start coding.

```
var rlmNet = new RlmNetwork("RLM_XOR_SAMPLE");
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 create 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 = 1;
rlmNet.StartRandomness = 30;
rlmNet.EndRandomness = 0;
```

The XOR app is a simple problem and one game should be enough to train it, thus, we have set the number of sessions to 1.

### 3.3. Starting the Session

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.

```
long sessionId = rlmNet.SessionStart();
```

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

```
var rnd = new Random();
for (int i = 0; i < 100; i++)
{
    //Code here for Processing the data, which includes, getting a random value
    //For both inputs, and score the RLM based on its output
}
```

### 3.4. Processing Your Data

We will need to code how we generate our random inputs.

```
// Input With Value List Declaration
var invs = new List<RlmIOWithValue>();

// get random value for Input1
string input1Value = Convert.ToBoolean(rnd.Next(0, 2)).ToString();
invs.Add(new RlmIOWithValue(rlmNet.Inputs.Where(item => item.Name ==
    "XORInput1").First(), input1Value));

// get random value for Input2
string input2Value = Convert.ToBoolean(rnd.Next(0, 2)).ToString();
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. Within this class is `RunCycle` method which



has return type of `rlm_cycleComplete_args`, 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);
```

Here we have a new Cycle made. This should be inside the loop that you made to iterate through all the sessions. This is the line of the block of code that trains the AI.

```
double score = ScoreCycle(result, input1Value, input2Value);
```

The first line of code here scores the RLM based on how you want it to be scored. This is a user made method and will depend on how the user decides on scoring the RLM's decision.

```
double score = 0;
string output = null;

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

foreach (var item in xorTable)
{
    if (item.Input1 == input1 && item.Input2 == input2)
    {
        if (item.Output == output)
        {
            score = 100;
        }

        break;
    }
}
```

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.

Finally, we have the code that saves it to our database, the `ScoreCycle` method of the `RlmNetwork` class.

```
rlmNet.ScoreCycle(result.CycleOutput.CycleID, score);
```

That concludes Developer's Guide for the RLM. Happy Coding!