For a better understanding of the code, here are some instructions.

The concept of this project is to create a tool capable of using all possible indicators in any given conditions, depending on how these conditions interact. Therefore, having a unified approach is crucial.

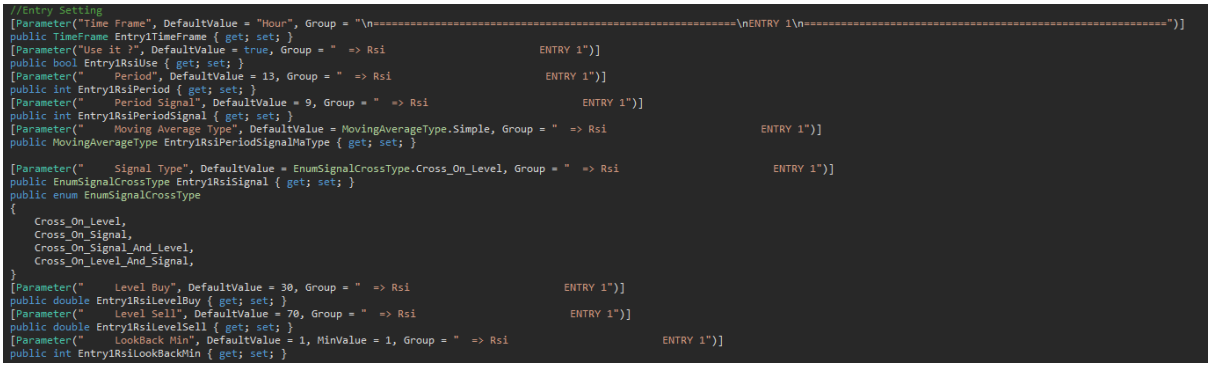
The NNFX strategy tester is divided into several parts:

1. Baseline
2. Entry
3. Confirmation
4. Volume
5. Exit

I.Here is the template for writing an indicator :

Each of these categories can be doubled to increase criteria, which is why each parameter is named accordingly.

1- Entry1 = categories name
2 -Entry1RsiUse = Indicator name
3 -Entry1Rsi(Use/Period/PeriodSignal etc..) Indicator Setting and Condition Critere name



```
//Entry Setting
[Parameter("Time Frame", DefaultValue = "Hour", Group = "\n=========================================================\nENTRY 1\n=========================================================")]
public TimeFrame Entry1TimeFrame { get; set; }
[Parameter("Use it ?", DefaultValue = true, Group = "  => Rsi                        ENTRY 1")]
public bool Entry1RsiUse { get; set; }
[Parameter("     Period", DefaultValue = 13, Group = "  => Rsi                         ENTRY 1")]
public int Entry1RsiPeriod { get; set; }
[Parameter("     Period Signal", DefaultValue = 9, Group = "  => Rsi                        ENTRY 1")]
public int Entry1RsiPeriodSignal { get; set; }
[Parameter("     Moving Average Type", DefaultValue = MovingAverageType.Simple, Group = "  => Rsi                        ENTRY 1")]
public MovingAverageType Entry1RsiPeriodSignalMaType { get; set; }

[Parameter("     Signal Type", DefaultValue = EnumSignalCrossType.Cross_On_Level, Group = "  => Rsi                        ENTRY 1")]
public EnumSignalCrossType Entry1RsiSignal { get; set; }
public enum EnumSignalCrossType
{
    Cross_On_Level,
    Cross_On_Signal,
    Cross_On_Signal_And_Level,
    Cross_On_Level_And_Signal,
}
[Parameter("     Level Buy", DefaultValue = 30, Group = "  => Rsi                        ENTRY 1")]
public double Entry1RsiLevelBuy { get; set; }
[Parameter("     Level Sell", DefaultValue = 70, Group = "  => Rsi                        ENTRY 1")]
public double Entry1RsiLevelSell { get; set; }
[Parameter("     LookBack Min", DefaultValue = 1, MinValue = 1, Group = "  => Rsi                        ENTRY 1")]
public int Entry1RsiLookBackMin { get; set; }
```

II.Here's how the code works:

The `Signal()` function calls all the robot's categories to determine the signal.

```
private int Signal(TradeType tradeType)
{
    // all function to get Signal Enter
    if ((GetBasline1(tradeType) == 1) && (GetEntry1(tradeType) == 1) && (GetCondirmation1(tradeType) == 1) && (GetVolume1(tradeType) == 1))
        return 1;

    else if ((GetBasline1(tradeType) == -1) && (GetEntry1(tradeType) == -1) && (GetCondirmation1(tradeType) == -1) && (GetVolume1(tradeType) == -1))
        return -1;

    else
        return 0;
}
```

The `get` functions call each specific category to determine if:

1. An indicator from this category is being used in the signal search, as it's possible not to use one or more categories.
2. The indicator(s) from this category have a signal that confirms the signal search.

```
private int GetBasline1(TradeType tradeType) // all indicator adding on baseline are implemented here
{
    //int idx1 = Baseline1TimeFrame == Chart.TimeFrame ? index : barsTF.OpenTimes.GetIndexByTime(Bars.OpenTimes[index]); -> need to backtest for knowing if special index is needed for multiTimeframe

    var resultMovingAverage = FunctionSignalUnderOver(Bars.ClosePrices, baseline1MovingAverage.Result, Baseline1MovingAverageSignal); // FunctionSignalUnderOver(DataSeries result, DataSeries Signal, EnumSignalOverUnderType signalOverUnder)

    // without baseline
    if (!Baseline1MovingAverageUse && tradeType == TradeType.Buy)
        return 1;
    else if (!Baseline1MovingAverageUse && tradeType == TradeType.Sell)
        return -1;
    // with baseline
    else if ((!Baseline1MovingAverageUse || resultMovingAverage == 1) && tradeType == TradeType.Buy) // add all variable with this sentence (exemple): (!Volume1MfiUse || resultMfi == 1)
        return 1;
    else if ((!Baseline1MovingAverageUse || resultMovingAverage == -1) && tradeType == TradeType.Sell)
        return -1;
    else
        return 0;
}
```

The `Function` determines the types of signals to search for. Currently, there are three types of functions:

1. Confirmation function (over/under)
2. Double confirmation function (over/under)
3. Signal function with a cross-determinant
4. Exit function with a cross-determinant

All other functions are either basic functions or statistical functions.

```
private int FunctionSignalDoubleSignalUnderOver(DataSeries result, DataSeries signal, EnumSignalDoubleOverUnderType signalOverUnder, double levelBuy, double levelSell)
{
    if (signalOverUnder == EnumSignalDoubleOverUnderType.Over_Level_And_Signal_Positive_Or_Under_Negative)
    {
        if (result.Last(1) > levelBuy && result.Last(1) > signal.Last(1))
            return 1;
        else if (result.Last(1) < levelSell && result.Last(1) < signal.Last(1))
            return -1;
        else
            return 0;
    }
    else if (signalOverUnder == EnumSignalDoubleOverUnderType.Over_Level_And_Signal_Positive_Or_Under_Negative)
    {
        if (result.Last(1) < levelBuy && result.Last(1) < signal.Last(1))
            return 1;
        else if (result.Last(1) > levelSell && result.Last(1) > signal.Last(1))
            return -1;
        else
            return 0;
    }
    else if (signalOverUnder == EnumSignalDoubleOverUnderType.Over_Level_And_Under_Signal_Positive_Or_Under_Level_And_Over_Signal_Negative)
    {
        if (result.Last(1) > levelBuy && result.Last(1) < signal.Last(1))
            return 1;
        else if (result.Last(1) < levelSell && result.Last(1) > signal.Last(1))
            return -1;
        else
            return 0;
    }

    else // Over_Level_And_Under_Signal_Negative_Or_Under_Level_And_Over_Signal_Positive
    {
        if (result.Last(1) < levelBuy && result.Last(1) > signal.Last(1))
            return 1;
        else if (result.Last(1) > levelSell && result.Last(1) < signal.Last(1))
            return -1;
        else
            return 0;
    }
}
// Real Enter Type Function -> min and max for having possibility to find range period with Multiple signal (Maybe need a special function) it work but it's not good when I
private int FunctionSignalCross(DataSeries result, DataSeries signal, EnumSignalCrossType signalOverUnder, double levelBuy, double levelSell, int lookBack)
{
    if (signalOverUnder == EnumSignalCrossType.Cross_On_Level)
    {
        if (result.Minimum(lookBack + 1) <= levelBuy && result.Last(1) > levelBuy)
            return 1;
        else if (result.Maximum(lookBack + 1) >= levelSell && result.Last(1) < levelSell)
            return -1;
        else
            return 0;
    }
    else if (signalOverUnder == EnumSignalCrossType.Cross_On_Level_And_Signal)
    {
        if (result.Minimum(lookBack + 1) <= levelBuy && result.Last(1) > levelBuy && signal.Last(1) > signal.Last(1))
            return 1;
        else if (result.Maximum(lookBack + 1) >= levelSell && result.Last(1) < levelSell && signal.Last(1) < signal.Last(1))
            return -1;
        else
            return 0;
    }
    else if (signalOverUnder == EnumSignalCrossType.Cross_On_Signal)
    {
        if (result.Minimum(lookBack + 1) <= signal.Last(2) && result.Last(1) > signal.Last(1))
            return 1;
        else if (result.Maximum(lookBack + 1) >= signal.Last(2) && result.Last(1) < signal.Last(1))
            return -1;
        else
            return 0;
    }
```

I wish you a pleasant read and a great collaboration on what promises to be an exceptional project.