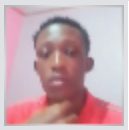
Larry Williams Market Secrets: Building a Swing Structure Indicator in MQL5

MetaTrader 5 — Indicators



Chacha Ian Maroa

Introduction

Financial markets move in waves. Prices rise and fall every day, and those movements can mark essential turning points. Larry Williams explains a straightforward mechanical way to read these points. With this understanding, we can build tools that highlight market structure and help traders decide when to enter or avoid trades.

In this article, we build a Market Structure Indicator in MQL5. It is based on Larry Williams' ideas of market swings. The indicator detects *short-term* swing points. It groups them into *intermediate-* and *long-term* points. This creates a clear picture of how the price is moving.

This article is part of a larger series on Larry Williams' market concepts. Each article in the series will explore one idea at a time and convert it into practical MQL5 tools. This keeps learning simple and helps you follow the logic step by step.

Who is Larry Williams?

Larry Williams is one of the most respected names in trading. He is a stock and commodity trader with a long track record. He is also the author of many trading books. One of his best-known releases is Long-Term Secrets to Short-Term Trading. Many traders study this book for its practical approach to market structure and swing analysis, which serves as the foundation for this article.

Larry Williams gained significant recognition after winning the World Cup Championship of Futures Trading in 1987. In that contest, he turned ten thousand dollars ($10000) into more than one million dollars ($1000000) within twelve months. No one has ever broken that record. Ten years later, his daughter Michelle Williams entered the same contest and won as well. This demonstrated that his ideas could be learned and applied successfully by others.

His work continues to influence modern technical trading. This makes him a perfect reference point for building structured tools such as this market structure indicator.

Understanding Williams' Market Structure Logic

Market structure, as taught by Larry Williams, is based on how price swings form over time. These swings appear naturally as the market moves up and down. When we understand these swings, we can follow the trend more confidently and judge whether the market is turning or continuing to move. This method removes guesswork and replaces it with a simple mechanical framework.

Larry Williams divides swing points into three groups. These are **short-term**, **intermediate-term**, and **long-term** swings. Each group is built from the one below it, allowing us to see the structure forming layer by layer. With a few rules, we can read charts in a clean, organized way.

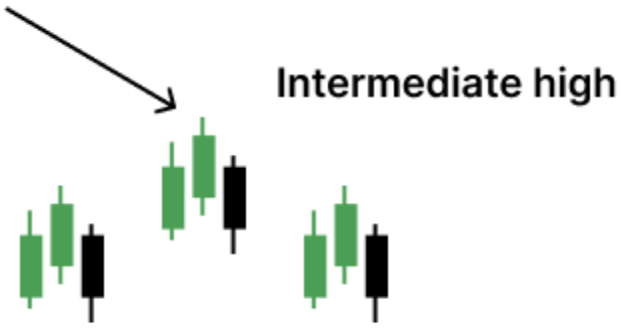
Short-term Swings

A short-term swing forms when the price completes a small move and turns back in the opposite direction. For example, a short-term low occurs when the price falls to a new low, fails to continue lower, and then rises again. In simple terms, the lowest bar sits between two bars with higher lows. This tells us that the downward move has finished for the moment.

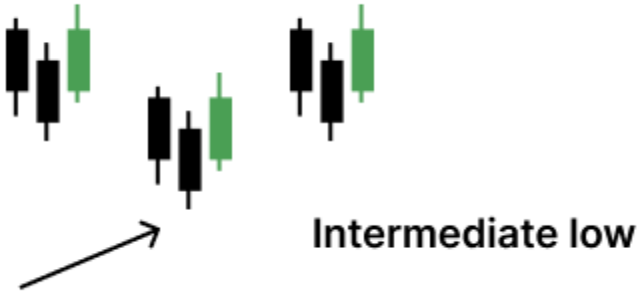
A short-term high forms in the opposite way. Price rises to a peak, then fails to continue higher and turns down instead. The middle high becomes a short-term high because the bars on both sides have lower highs. Short-term swings mark the most minor turning points, but they are the foundation of everything else we build later.

Intermediate-term Swings

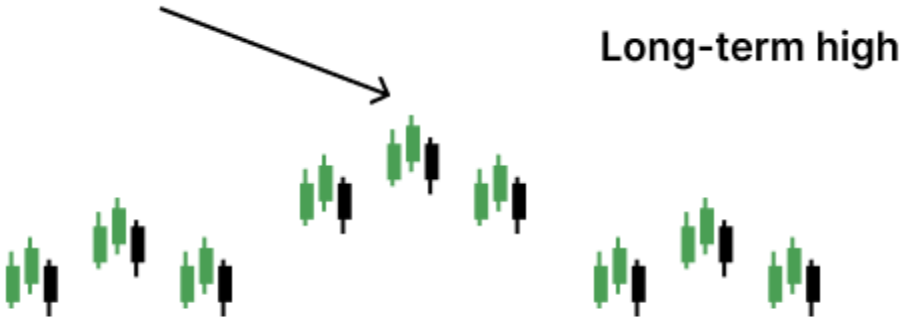
When enough short-term swings form, we can identify a swing of the next level. An intermediate-term high is a short-term high that stands above two other short-term highs.



Likewise, an intermediate-term low forms when higher short-term lows on both sides follow a short-term low.



These swings represent a more significant turning point because they confirm that the price has changed direction longer than a normal short-term fluctuation.
This level of swings allows us to remove slight noise and see the structure more clearly. The chart begins to form waves that are easier to track. The market no longer looks random because we can see a sequence: short-term highs and lows combine to form intermediate swings.
Long-term Swings
Once intermediate swings begin to form, we can step one level higher. A long-term swing happens when an intermediate-term swing stands above or below other intermediate swings around it. A long-term high appears when lower intermediate highs on both sides follow an intermediate high.



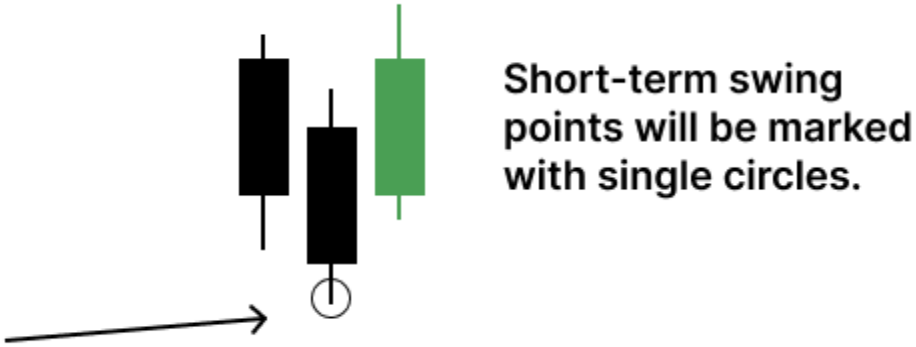
A long-term low forms when surrounding intermediate lows sit above it.

These long-term swings often show major turning points in the market.
When all three levels are visible, the chart resembles a staircase. First, you notice small steps created by short-term swings. Several of these combine into an intermediate swing. And several intermediate swings form a long-term swing. This nested structure reveals trend direction, trend strength, and possible reversal points with great clarity. This is why Larry Williams focuses heavily on swing logic.

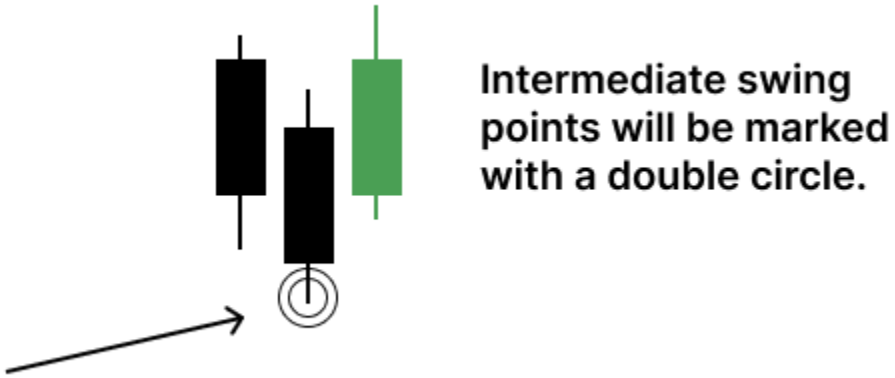
Indicator Design and Visual Structure

Before we write any code, we need to understand how this indicator will appear when loaded on a chart. The goal is not only to detect market structure but to present it in a way that is visually meaningful and

easy to read. Traders should be able to glance at the chart and instantly recognize market turns without struggling to interpret raw values or hidden logic.
This indicator will display three different levels of market structure. These levels will be drawn differently so that the trader can tell them apart without confusion.
1. Short-term swing points
Short-term turning points will be marked with single circles.



These are the smallest swings and occur most frequently. They help us read the market's rhythm one step at a time. When the indicator finds a short-term low, it will place a circle below the candle. When it finds a short-term high, another circle will appear above the candle.
2. Intermediate swing points
Intermediate swings grow out of short-term swings. They are more meaningful because they occur less often and represent a shift in market pressure. These points will be marked using a double circle, a smaller ring within a larger one.



The outer ring shows that a turning point has formed, while the inner ring confirms its significance. When a trader sees this symbol, they know the price has turned with more momentum than a typical short-term swing.
3. Long-term swing points
The final and most decisive category is the long-term swing. These are the major turning points that shape the larger trend. They may take time to form, but once they do, they matter the most. Long-term swing highs and long-term swing lows will be marked using arrows.

Long-term swing points will be marked using arrows

An arrow above the price signals a strong swing high and possible downward movement. An arrow below the price signals a strong swing low and possible upward movement. When these arrows appear, they stand out clearly and draw immediate attention.

Implementing the Market Structure Logic in MQL5

In this section, we begin turning the market structure logic into a real MQL5 indicator. MQL5 programs come in four types: Scripts, Indicators, Expert Advisors, and Services. Because our goal is to detect and draw swing structure on the chart, we will implement this as a custom indicator.

To start, open MetaEditor 5, create a new indicator file, and name it:

`larryWilliamsMarketStructureIndicator.mq5`

Once the file is created, delete everything inside it and paste in the following starter code:

```
//+------------------------------------------------------------------+
//|                        larryWilliamsMarketStructureIndicator.mq5 |
//|          Copyright 2025, MetaQuotes Ltd. Developer is Chacha Ian |
//|                           https://www.mql5.com/en/users/chachaian |
//+------------------------------------------------------------------+

#property copyright "Copyright 2025, MetaQuotes Ltd. Developer is Chacha Ian"
#property link      "https://www.mql5.com/en/users/chachaian"
#property version   "1.00"

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){

   return(INIT_SUCCEEDED);
}

//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int32_t  rates_total,
                const int32_t  prev_calculated,
                const datetime &time [],
                const double   &open [],
                const double   &high [],
                const double   &low  [],
                const double   &close[],
                const long     &tick_volume[],
                const long     &volume[],
                const int32_t  &spread[])
{
   //--- Return total bars processed
   return(rates_total);
}
```

```
//--- UTILITY FUNCTIONS

//+------------------------------------------------------------------+
```

Breaking down the code

Below is what each part of the file means and why it matters:

1. File Header

```
//+------------------------------------------------------------------+
//|                        larryWilliamsMarketStructureIndicator.mq5 |
//|          Copyright 2025, MetaQuotes Ltd. Developer is Chacha Ian |
//|                            https://www.mql5.com/en/users/chachaian |
//+------------------------------------------------------------------+
```

The first comment block at the top is only for readability. It holds the file name, author details, and copyright notice. It does not affect how the indicator works — it simply documents ownership.

2. Property declarations

The *#property* lines that follow set metadata for the indicator.

```
#property copyright "Copyright 2025, MetaQuotes Ltd. Developer is Chacha Ian"
#property link      "https://www.mql5.com/en/users/chachaian"
#property version   "1.00"
```

Here we define the copyright, link, and version number. More properties will be added later to control chart drawings and buffer plots.

3. *OnInit* function

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){

   return(INIT_SUCCEEDED);
}
```

This function is executed only once, right at the moment the indicator starts. We will later use this section to allocate buffers, configure plots, and set drawing styles. For now, we return success.

4. *OnCalculate* function

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int32_t  rates_total,
                const int32_t  prev_calculated,
                const datetime &time [],
                const double   &open [],
                const double   &high [],
                const double   &low  [],
                const double   &close[],
                const long     &tick_volume[],
                const long     &volume[],
                const int32_t  &spread[])
{
   //--- Return total bars processed
   return(rates_total);
}
```

This function runs every time the chart updates or a new candle forms. It receives price arrays (Open, High, Low, Close) and returns the number of bars processed. Soon, we will add all swing-detection logic here—short-term, intermediate, and long-term.

5. Utility functions section

At the bottom is a placeholder for helper functions. This is where we will place our utility functions. Keeping utility functions separate makes the main code cleaner and easier to read.

Now that the project file is in place, the next step is to tell MetaTrader 5 where our indicator should appear. Every indicator must indicate whether it will be displayed in the main price chart or in a separate

sub-window. Sub-window indicators include tools such as RSI and MACD, while price-based indicators, such as moving averages, are usually plotted directly on the chart.

In our case, the swing structure we are building is price-based. The turning points must align with the candles; therefore, they must be drawn on the main chart window. MQL5 allows us to specify this using a simple directive.

Add the following section just below the property lines at the top of the file:

```
//+------------------------------------------------------------------+
//| Custom Indicator specific directives                             |
//+------------------------------------------------------------------+
#property indicator_chart_window
```

This line instructs MetaTrader 5 to display the output of our indicator on the main candle chart rather than in a separate sub-window. Once this is set, every graphic element we draw will attach directly to price bars, which is precisely what we want for identifying swing highs and lows.

With the display location set, we now define how many plots and buffers our indicator will use. In MQL5, every visual output drawn on the chart is linked to an indicator buffer. A buffer is an array that stores price values or signals that later appear as shapes or lines. The number of plots we declare tells MetaTrader 5 how many independent visual elements we intend to draw.

For our market structure indicator, we have six possible swing points, each with its own buffer. The six swing types we will detect are:

1. Short-term low
2. Short-term high
3. Intermediate-term low
4. Intermediate-term high
5. Long-term low
6. Long-term high

Each of these points will be drawn separately on the chart. Since one plot requires one buffer, we need six buffers and six plots. Add the following code under the existing property directives:

```
//+------------------------------------------------------------------+
//| Custom Indicator specific directives                             |
//+------------------------------------------------------------------+

...

#property indicator_plots 6
#property indicator_buffers 6
```

These two properties tell MetaTrader 5 that the indicator will produce six graphical outputs and that memory must be allocated for six buffers to store those values.

Now that we have specified the total number of buffers our indicator will use, the next step is to declare the actual storage arrays. These arrays will hold the values for the first category of swing points: short-term highs and short-term lows. We place them in the global area of the file so that they are accessible throughout the indicator. Add the following lines just below the property directives:

```
//+------------------------------------------------------------------+
//| Indicator buffers                                                |
//+------------------------------------------------------------------+
double shortTermLows [];
double shortTermHighs[];
```

At this stage, the arrays do not yet draw anything. They serve only as containers for swing point values.

After declaring our storage arrays in the global scope, the next logical task is to register them as indicator buffers. This step allows MetaTrader 5 to recognize the arrays as part of the drawing system. Once registered, any values stored inside them can be plotted in real time on the chart.

We perform this registration in the OnInit function using SetIndexBuffer. This function links a specific array to a buffer index inside the indicator. The platform will later use this link to draw our swing points.

Add the following code inside the *OnInit* function:

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){
```

```
   //--- Bind arrays to indicator buffers
   SetIndexBuffer(0, shortTermLows,  INDICATOR_DATA);
   SetIndexBuffer(1, shortTermHighs, INDICATOR_DATA);

   return(INIT_SUCCEEDED);
}
```

The next step is to build the algorithm that detects the short-term swing points. This logic will be placed inside the *OnCalculate* function. This is where the indicator does its work. MetaTrader 5 calls it whenever the chart updates. This includes when you first attach the indicator, when a new candle forms, and when price ticks arrive.

Two situations matter most. First is the initial run. You detect it when **prev_calculated equals zero**. Use this case to prepare the indicator. For example, initialize buffers, clear any previous values, and fill historical swings if you want full history shown at load.

Second is the new bar event. This is when **prev_calculated is less than rates_total**. In that case, you should be able to implement logic that runs once per closed candle. Doing this prevents repeated work on every tick and keeps the indicator efficient.

Could you now update your OnCalculate function to match the structure shown below?

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int32_t  rates_total,
                const int32_t  prev_calculated,
                const datetime &time [],
                const double   &open [],
                const double   &high [],
                const double   &low  [],
                const double   &close[],
                const long     &tick_volume[],
                const long     &volume[],
                const int32_t  &spread[])
{
   //--- Start with clean buffers at first calculation
   if(prev_calculated == 0){
   }

   //--- Recalculate structures only when a new bar is added
   if(prev_calculated < rates_total){
   }

   //--- Return total bars processed
   return(rates_total);
}
```

When the indicator is launched for the first time, we must make sure that all our buffers start with clean values. This prevents the arrays from holding any leftover data and also ensures that we only display swing points that actually exist. Since swing signals do not appear on every bar, it is essential to start with empty buffers and then fill them with fresh values during the calculation process.

To achieve this, we add the initialization logic inside the block that checks if prev_calculated == 0. This block runs only once when the indicator is first attached to the chart. Here, we use the ArrayInitialize function to fill both arrays with EMPTY_VALUE. EMPTY_VALUE is a special value that tells MetaTrader 5 not to draw anything for that bar.

Update your code to match the structure below:

```
//--- Start with clean buffers at first calculation
   if(prev_calculated == 0){

      ArrayInitialize(shortTermLows,  EMPTY_VALUE);
      ArrayInitialize(shortTermHighs, EMPTY_VALUE);

   }
```

This simple step prepares the indicator for the next stage, where we start detecting and plotting short-term swing points.

Detecting Short-Term Swing Points in *OnCalculate*

```
  //--- Recalculate structures only when a new bar is added
   if(prev_calculated < rates_total){

      ArrayInitialize(shortTermLows,  EMPTY_VALUE);
      ArrayInitialize(shortTermHighs, EMPTY_VALUE);

      for(int32_t i = 1; i < rates_total - 2; i++){
      //--- Identify a short-term low
      if(low[i] < low[i - 1] && low[i] < low[i + 1]){
            shortTermLows[i] = low[i];
      }

      //--- Identify a short-term high
      if(high[i] > high[i - 1] && high[i] > high[i + 1]){
            shortTermHighs[i] = high[i];
      }
      }
   }
```

This block runs when the indicator is first launched and whenever a new bar appears. We clear the short-term buffers, then scan the price history to identify short-term highs and lows. The code uses a simple three-bar rule. A short-term low is a bar whose low is lower than the low of the bar to its left and lower than the low of the bar to its right. A short-term high is the opposite. Its height is higher than the height of the left bar and higher than the height of the right bar.

Below is what each part does and why it is needed

```
ArrayInitialize(shortTermLows,  EMPTY_VALUE);
ArrayInitialize(shortTermHighs, EMPTY_VALUE);
```

These calls reset both buffers to empty values. The EMPTY_VALUE constant instructs MetaTrader 5 not to display anything for the bar. Clearing the arrays ensures that old marks do not persist, allowing us to start with a clean slate for the current pass.

```
for(int32_t i = 1; i < rates_total - 2; i++){
}
```

The loop walks through each usable bar index. We start at one because we need a left neighbor at index i-1. We end before the last bars so that a right neighbor at i+1 always exists. This prevents out-of-range reads and avoids errors.

Inside the loop, we apply the three-bar tests.

```
if(low[i] < low[i - 1] && low[i] < low[i + 1]){
    shortTermLows[i] = low[i];
 }
```

This checks for a short-term low. If the condition is *true*, we write the low price into the *shortTermLows* buffer at index i. Writing a real price into the buffer makes the plotting system draw the symbol for that bar.

```
//--- Identify a short-term high
if(high[i] > high[i - 1] && high[i] > high[i + 1]){
   shortTermHighs[i] = high[i];
}
```

This checks for a short-term high. If it passes, we write the high price into the *shortTermHighs* buffer at index i. Writing the value makes the chart show the high marker for that bar.

Reinitializing and recalculating when a new candle forms makes the indicator reflect the current structure. A swing can appear or vanish when new price data arrives. Resetting the buffers removes any stale or incorrect marks. This keeps the visible structure correct and consistent.

Setting Up Graphic Plots for Short-Term Swing Points

At this stage, our indicator can detect short-term highs and lows, and the values already appear in the Data Window. However, when you attach the indicator to a price chart, nothing is visible on the chart yet. This is

normal. In MQL5, an indicator only becomes visible after we configure one or more graphic plots. A plot tells MetaTrader 5 how to draw the data stored in a specific indicator buffer. Without this step, the platform does not know what shapes to draw or how they should look.

We will add two plots. One plot will show short-term lows, and the other will show short-term highs. Both plots will use a circular Wingdings symbol. Larry Williams often describes these points as “ringed” swings, so the **circular symbol** is a perfect match.

You can place the following code inside the *OnInit* function, directly after registering the indicator buffers: Setting Up Graphic Plots for Short-Term Swing Points.

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){
   ...

   //--- Configure Graphic Plots
   PlotIndexSetInteger(0, PLOT_ARROW, 161);
   PlotIndexSetDouble (0, PLOT_EMPTY_VALUE, EMPTY_VALUE);
   PlotIndexSetString (0, PLOT_LABEL, "ShortTermLows");

   PlotIndexSetInteger(1, PLOT_DRAW_TYPE, DRAW_ARROW);
   PlotIndexSetInteger(1, PLOT_ARROW, 161);
   PlotIndexSetDouble (1, PLOT_EMPTY_VALUE, EMPTY_VALUE);
   PlotIndexSetString (1, PLOT_LABEL, "ShortTermHighs");

   return(INIT_SUCCEEDED);
}
```

Let us break down what each line does.

```
PlotIndexSetInteger(0, PLOT_DRAW_TYPE, DRAW_ARROW);

...

PlotIndexSetInteger(1, PLOT_DRAW_TYPE, DRAW_ARROW);
```

This sets the drawing style. We choose DRAW_ARROW because circular Wingdings symbols are part of the arrow family. This allows us to draw a small ring on each bar where a swing occurs.

```
PlotIndexSetInteger(0, PLOT_ARROW, 161);

...

PlotIndexSetInteger(1, PLOT_ARROW, 161);
```

Here, we select the Wingdings character code. Code 161 is a circular symbol that looks like a clean ring on the chart. This matches the visual style Larry Williams uses for swing points.

```
PlotIndexSetDouble (0, PLOT_EMPTY_VALUE, EMPTY_VALUE);

...

PlotIndexSetDouble (1, PLOT_EMPTY_VALUE, EMPTY_VALUE);
```

This tells MetaTrader 5 what “do not draw anything” means. Since we initialize unused bars with EMPTY_VALUE, the indicator will only draw a symbol on bars where we write a real price into the buffer.

```
PlotIndexSetString (0, PLOT_LABEL, "ShortTermLows");

...

PlotIndexSetString (1, PLOT_LABEL, "ShortTermHighs");
```

This gives each plot a label. These labels appear in the Data Window so you can easily identify which series corresponds to which type of swing point.

To complete the setup, we must also control the plot appearance using *property directives*. Add these lines just below the existing *#property* directives:

```
#property indicator_color1 clrGreen
#property indicator_color2 clrBlack

#property indicator_width1 1
#property indicator_width2 1
```

Here is what these directives do:

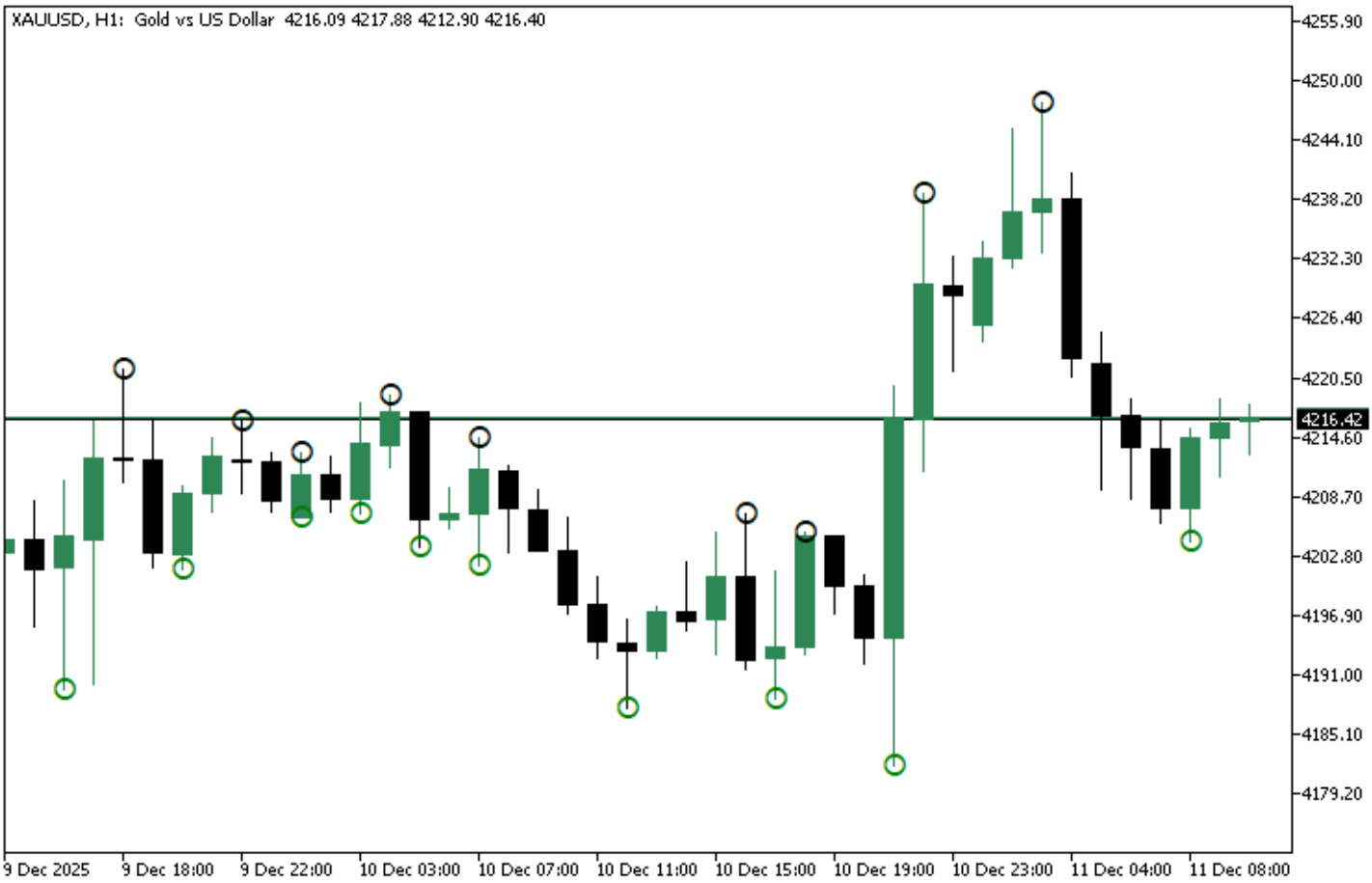
indicator_color1 and indicator_color2

These set the colors for the two plots. The first color indicates short-term lows, and the second indicates short-term highs. You can adjust these later if you'd like a different color scheme.

indicator_width1 and indicator_width2

These set the thickness of the plot symbols. Since we use circular Wingdings characters, width 1 keeps them clean and easy to read.

At this point, the indicator has basic detection logic, buffer registration, and plot configuration. It is ready for a first test. Save your file, press **Compile** in MetaEditor 5, and make sure there are no errors. Then attach the indicator to any chart. Use a timeframe with enough bars, like daily or H1. Once loaded, watch how small circles appear on the chart. These circles mark the short-term swing lows and highs. Scroll through history or wait for a new bar to form — you should see the swing points update automatically. These set the thickness of the plot symbols. Since we use circular Wingdings characters, width 1 keeps them clean and easy to read.



Building Intermediate Swing Points

Now that we can detect short-term swing points, the next goal is to build intermediate swing points from them. Larry Williams teaches that market structure forms in layers. Short-term swings form the foundation, and from these we can derive larger swings that give a clearer picture of trend strength. To support this next phase, we begin by declaring two more global arrays. One will store intermediate swing lows, and the other will store intermediate swing highs.

```
double intermediateTermLows [];
double intermediateTermHighs[];
```

Inside the *OnInit* function, we link these arrays to new indicator buffer indices by calling *SetIndexBuffer*. This makes sure MetaTrader 5 knows where the values for these plots will come from.

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){

   ...

   SetIndexBuffer(2, intermediateTermLows,  INDICATOR_DATA);
   SetIndexBuffer(3, intermediateTermHighs, INDICATOR_DATA);

   ...

   return(INIT_SUCCEEDED);
}
```

Once that is in place, we can introduce two small utility functions. Their purpose is simple. They scan the short-term swing points we already calculated and extract only the points that qualify as intermediate swings.

```
//+------------------------------------------------------------------+
//| Build intermediate highs from short-term highs                   |
//+------------------------------------------------------------------+
void BuildIntermediateHighs(const double &shortHighs[], const int32_t rates_total, double
&intermediateHighs[])
{
   // ensure target array sized
   if(ArraySize(intermediateHighs) != rates_total) ArrayResize(intermediateHighs,
rates_total);

   // clear intermediate buffer
   for(int32_t i = 0; i < rates_total; ++i) intermediateHighs[i] = EMPTY_VALUE;

   // collect indices of short-term highs
   int SH_idx[];
   ArrayResize(SH_idx, 0);
   for(int32_t i = 0; i < rates_total; ++i)
   {
      if(shortHighs[i] != EMPTY_VALUE)
      {
      int newlen = ArraySize(SH_idx) + 1;
      ArrayResize(SH_idx, newlen);
      SH_idx[newlen - 1] = i;
      }
   }

   // compress: each short high with lower short highs on both sides becomes an intermediate
high
   int count = ArraySize(SH_idx);
   if(count < 3) return; // need at least three short-highs for a middle one to qualify

   for(int k = 1; k < count - 1; ++k)
   {
      int prev_i = SH_idx[k - 1];
      int cur_i  = SH_idx[k];
      int next_i = SH_idx[k + 1];

      // strict comparison per Larry: current must be higher than neighbors
      if(shortHighs[cur_i] > shortHighs[prev_i] && shortHighs[cur_i] > shortHighs[next_i])
      intermediateHighs[cur_i] = shortHighs[cur_i];
   }
}

//+------------------------------------------------------------------+
//| Build intermediate lows from short-term lows                     |
```

```
//+------------------------------------------------------------------+
void BuildIntermediateLows(const double &shortLows[], const int32_t rates_total, double
&intermediateLows[])
{
   if(ArraySize(intermediateLows) != rates_total) ArrayResize(intermediateLows,
rates_total);

   for(int32_t i = 0; i < rates_total; ++i) intermediateLows[i] = EMPTY_VALUE;

   int SL_idx[];
   ArrayResize(SL_idx, 0);
   for(int32_t i = 0; i < rates_total; ++i)
   {
      if(shortLows[i] != EMPTY_VALUE)
      {
      int newlen = ArraySize(SL_idx) + 1;
      ArrayResize(SL_idx, newlen);
      SL_idx[newlen - 1] = i;
      }
   }

   int count = ArraySize(SL_idx);
   if(count < 3) return;

   for(int k = 1; k < count - 1; ++k)
   {
      int prev_i = SL_idx[k - 1];
      int cur_i  = SL_idx[k];
      int next_i = SL_idx[k + 1];

      // strict comparison: current low must be lower than neighbors
      if(shortLows[cur_i] < shortLows[prev_i] && shortLows[cur_i] < shortLows[next_i])
      intermediateLows[cur_i] = shortLows[cur_i];
   }
}
```

Let us use the function *BuildIntermediateHighs* to explain how these utilities work. The function begins by ensuring the target array has enough space, then clears it by filling it with empty values. It then loops through the short-term highs and collects the bar indices where real swing highs occur. These indices are stored in a small temporary list. Once we have this list, the function checks each short-term high in the middle of its neighbors. A bar qualifies as an intermediate swing high only when its value is higher than the short-term high before it and higher than the one after it. This is the classic three-point structure. A left shoulder, a higher middle peak, and a right shoulder that is lower. When this condition is true, the value is written back to the intermediate highs buffer.

The function that builds intermediate lows follows the same steps. The only difference is that it checks for lower values instead of higher ones. So while one function looks for a middle peak, the other looks for a middle trough. Apart from this inverse comparison, the overall logic is identical.

After creating the utility functions that build intermediate swing points, the next step is to integrate them into the indicator's workflow. Just as with short-term buffers, intermediate swing buffers must start in a clean state. We achieve this by initializing them to EMPTY_VALUE when the indicator is first launched on the chart. This happens in the *OnCalculate* function, in the section that runs when **prev_calculated equals zero**.

```
//--- Start with clean buffers at first calculation
if(prev_calculated == 0){

   ...

   ArrayInitialize(intermediateTermLows,  EMPTY_VALUE);
   ArrayInitialize(intermediateTermHighs, EMPTY_VALUE);

}
```

Once the initialization is done, we can compute the intermediate swing points. This is performed inside the block that runs when a new bar forms or when the indicator is loaded for the first time. In other words, this happens when **prev_calculated is less than rates_total**. We already use this block to calculate short-term swing points, so we add our function calls below that section.

```
//--- Recalculate structures only when a new bar is added
if(prev_calculated < rates_total){

   ...

   BuildIntermediateLows(shortTermLows, rates_total, intermediateTermLows);
   BuildIntermediateHighs(shortTermHighs, rates_total, intermediateTermHighs);

}
```

These two calls allow the indicator to derive intermediate swings directly from the updated short-term structure. Each time a new bar is added, the indicator recalculates the foundation layer first and then applies the second layer. This creates a clean and consistent hierarchy of swing points that updates smoothly as the market moves.

With the intermediate swing logic now entirely in place, the next step is to make these values visible on the chart. The calculation alone is not enough. Just as with the short-term structures, the intermediate swing points require their own graphic plot configuration within the *OnInit* function. We add the following two plot definitions directly below the existing short-term plot settings.

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){

   ...

   PlotIndexSetInteger(2, PLOT_DRAW_TYPE, DRAW_ARROW);
   PlotIndexSetInteger(2, PLOT_ARROW, 161);
   PlotIndexSetDouble (2, PLOT_EMPTY_VALUE, EMPTY_VALUE);
   PlotIndexSetString (2, PLOT_LABEL, "intermediateTermLows");

   PlotIndexSetInteger(3, PLOT_DRAW_TYPE, DRAW_ARROW);
   PlotIndexSetInteger(3, PLOT_ARROW, 161);
   PlotIndexSetDouble (3, PLOT_EMPTY_VALUE, EMPTY_VALUE);
   PlotIndexSetString (3, PLOT_LABEL, "IntermediateTermHighs");

   return(INIT_SUCCEEDED);
}
```

These lines follow the same structure as the earlier configurations, so their behavior should already be familiar. They instruct the terminal how to draw the values stored in the intermediate swing buffers, using the same circular Wingdings character as the short-term points. The difference becomes clear when we introduce the display properties.

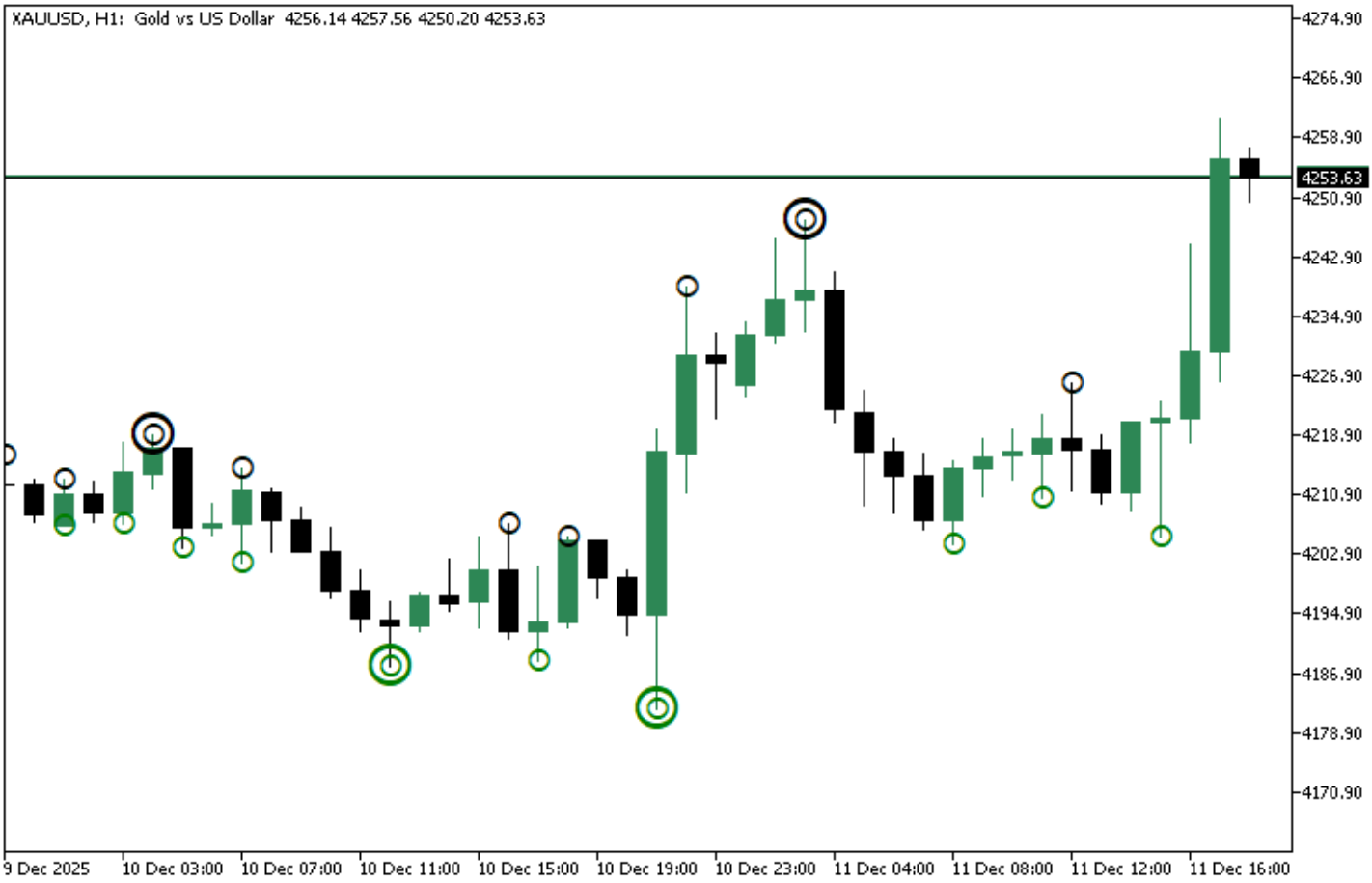
To help the trader visually distinguish intermediate swings from the short-term ones, we define an additional set of property directives:

```
#property indicator_color3 clrGreen
#property indicator_color4 clrBlack

#property indicator_width3 4
#property indicator_width4 4
```

Here we use the same colors as before, but significantly increase the width. As a result, the intermediate swing points appear as larger circular marks. When an intermediate swing occurs at the same bar as a short-term swing, the larger circle naturally sits over the smaller one, creating a double-ringed effect. This is intentional and follows Larry Williams' convention of making higher-level swing points more visually dominant on the chart.

At this stage, you can recompile the indicator and attach it to any chart, such as the one-hour or daily timeframe. If everything has been typed correctly, you should now see both short-term and intermediate

swing points displayed clearly on the screen. This is a good time to make sure your logic is working as expected and that the indicator correctly identifies the different levels of market structure.



Building Long-term Swing Points

With the short- and intermediate-term structures already in place, the final layer of our market structure hierarchy is the long-term swing points. These long-term swings are derived from the intermediate swings in the same way intermediate swings were derived from short-term swings. Since the reader is familiar with this workflow, we will focus only on the new parts introduced at this level.

Just as before, we begin by declaring two arrays in the global scope. These arrays will hold the values of long-term lows and highs, which will later be mapped to their own indicator buffers:

```
double longTermLows [];
double longTermHighs[];
```

Inside the *OnInit* function, we register these arrays as indicator buffers so that the terminal knows they will hold plot data:

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){

   ...

   SetIndexBuffer(4, longTermLows,  INDICATOR_DATA);
   SetIndexBuffer(5, longTermHighs, INDICATOR_DATA);

   ...

   return(INIT_SUCCEEDED);
}
```

When the indicator runs for the first time, we prepare these buffers by filling them with EMPTY_VALUE inside the **prev_calculated == 0** block. This ensures that only actual long-term swing points will be plotted:

```
//--- Start with clean buffers at first calculation
if(prev_calculated == 0){

   ...

   ArrayInitialize(longTermLows,  EMPTY_VALUE);
   ArrayInitialize(longTermHighs, EMPTY_VALUE);

}
```

The logic for deriving the long-term swing points closely mirrors what we have already implemented. We are still using the same builder functions—*BuildIntermediateLows* and *BuildIntermediateHighs*. The only difference is that at this stage, instead of passing short-term swings as inputs, we pass intermediate-term swings. This allows the same algorithm to identify long-term pivots using the next level of data. Inside the block that handles recalculation, the calls look like this:

```
//--- Recalculate structures only when a new bar is added
if(prev_calculated < rates_total){

   ...

   BuildIntermediateLows(intermediateTermLows, rates_total, longTermLows);
   BuildIntermediateHighs(intermediateTermHighs, rates_total, longTermHighs);

}
```

With the long-term swing points computed, the next step is to configure their graphical appearance. These points should stand out clearly on the chart, so instead of using circular Wingdings (as we did for short-term and intermediate swings), we draw them using arrow symbols. We also shift the arrows slightly above or below the price so they do not overlap with candles or previous swing points. This visual separation helps the reader instantly distinguish long-term structure from the rest.

Inside the *OnInit* function, configure the plots as follows:

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){

   ...

   PlotIndexSetInteger(4, PLOT_DRAW_TYPE, DRAW_ARROW);
   PlotIndexSetInteger(4, PLOT_ARROW, 233);
   PlotIndexSetInteger(4, PLOT_ARROW_SHIFT, +30);
   PlotIndexSetDouble (4, PLOT_EMPTY_VALUE, EMPTY_VALUE);
   PlotIndexSetString (4, PLOT_LABEL, "LongTermHighs");

   PlotIndexSetInteger(5, PLOT_DRAW_TYPE, DRAW_ARROW);
   PlotIndexSetInteger(5, PLOT_ARROW, 234);
   PlotIndexSetInteger(5, PLOT_ARROW_SHIFT, -30);
   PlotIndexSetDouble (5, PLOT_EMPTY_VALUE, EMPTY_VALUE);
   PlotIndexSetString (5, PLOT_LABEL, "LongTermHighs");

   return(INIT_SUCCEEDED);
}
```

To complete the visual configuration, we add two more *#property* directives to assign colors and line widths appropriate for long-term swings:

```
#property indicator_color5 clrGreen
#property indicator_color6 clrBlack

#property indicator_width5 2
  #property indicator_width6 2
```

Before we conclude the implementation, there is one more improvement worth adding. Since this indicator relies heavily on visual interpretation of swing points, it helps a great deal if the chart itself is clean,

consistent, and easy to read. Different users may have different chart templates loaded, and some of those templates—dark themes, heavy gridlines, exotic candle colors—can make the plotted swing structure difficult to see.

To ensure that the indicator always displays clearly, we define a small utility function, *ConfigureChartAppearance*.

```
//+------------------------------------------------------------------+
//| This function configures the chart's appearance.                 |
//+------------------------------------------------------------------+
bool ConfigureChartAppearance()
{
   if(!ChartSetInteger(0, CHART_COLOR_BACKGROUND, clrWhite)){
      Print("Error while setting chart background, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_SHOW_GRID, false)){
      Print("Error while setting chart grid, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_MODE, CHART_CANDLES)){
      Print("Error while setting chart mode, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_COLOR_FOREGROUND, clrBlack)){
      Print("Error while setting chart foreground, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_COLOR_CANDLE_BULL, clrSeaGreen)){
      Print("Error while setting bullish candles color, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_COLOR_CANDLE_BEAR, clrBlack)){
      Print("Error while setting bearish candles color, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_COLOR_CHART_UP, clrSeaGreen)){
      Print("Error while setting bearish candles color, ", GetLastError());
      return false;
   }

   if(!ChartSetInteger(0, CHART_COLOR_CHART_DOWN, clrBlack)){
      Print("Error while setting bearish candles color, ", GetLastError());
      return false;
   }

   return true;
}
```

This function adjusts the chart to a simple, high-contrast layout by setting a white background, turning off the grid, and applying consistent foreground and candle colors. We are not modifying any trading logic here; the function only standardizes the chart so that the swing plots stand out immediately when the indicator is loaded.

Once the function is defined, we call it at the very beginning of the *OnInit* function.

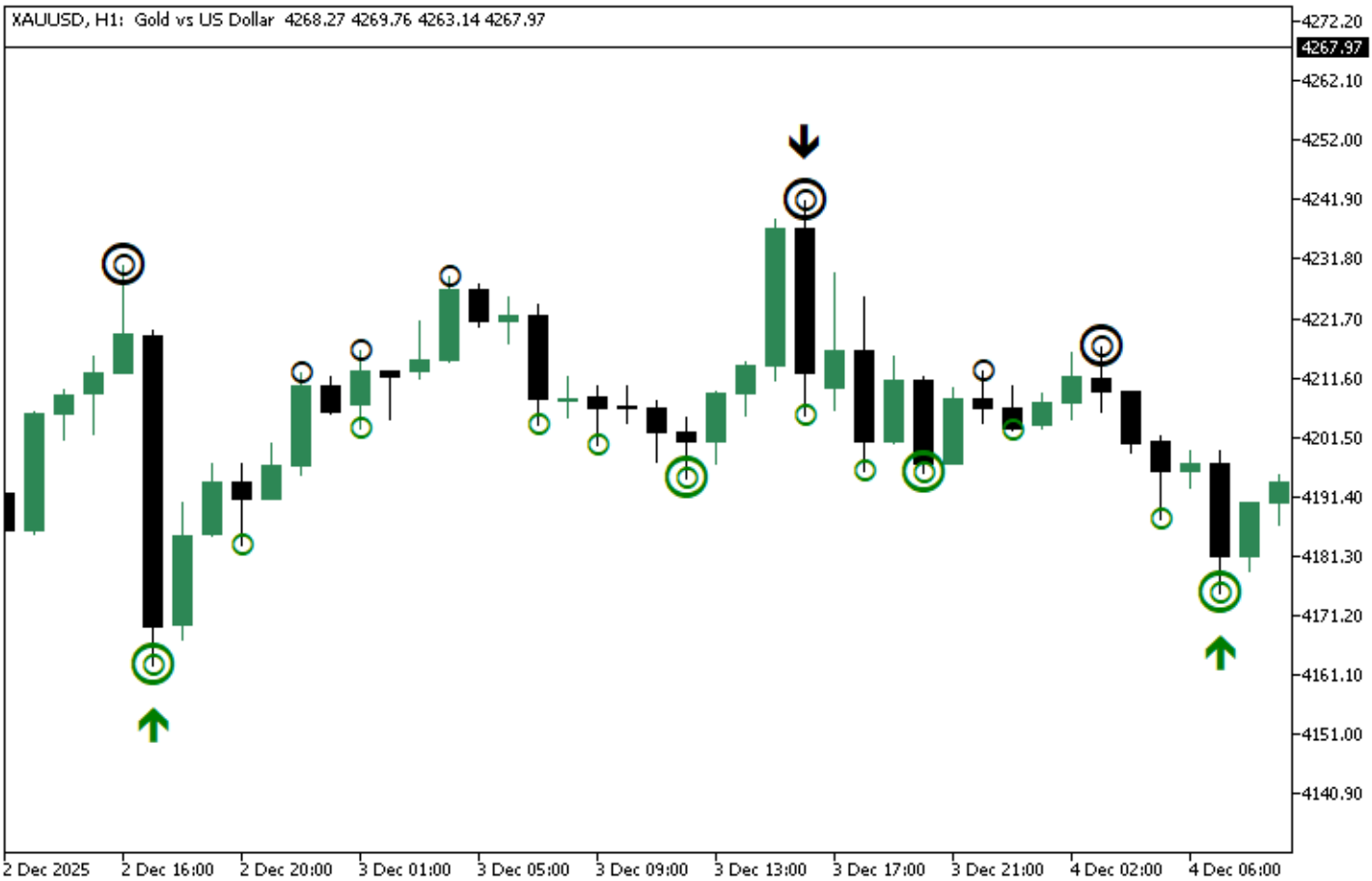
```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(){
```

```
   //--- To configure the chart's appearance
   if(!ConfigureChartAppearance()){
      Print("Error while configuring chart appearance", GetLastError());
      return INIT_FAILED;
   }

   ...

}
```

If the configuration is applied successfully, the indicator proceeds with its normal initialization. Otherwise, an error is reported. Placing this call at the top ensures that the chart is fully prepared before any buffers or plots are set up.

With this final addition, the indicator not only computes structure reliably but also presents it in a clean, professional visual layout, making the underlying swing levels easy to identify at a glance.

You can go ahead and recompile the indicator, then launch it on a chart. With all three layers implemented, the entire swing structure—short-term, intermediate-term, and long-term should now be visible and easy to distinguish at a glance.



With this final touch, our indicator is complete. All components—from identifying short-term, intermediate-term, and long-term swing points to configuring the chart for clear visualization—are now fully implemented. To make your work easier, the complete source code has been attached as "*larryWilliamsMarketStructureIndicator.mq5*." If at any point you feel that something is missing or not working as expected, compare your implementation with the attached reference file.

How to Use the Indicator in Your Trading

Now that the indicator is fully functional and displays short-term and intermediate swing points on the chart, it is essential to explain how a trader can actually use this information. Market structure is one of the most valuable tools in price analysis, and these swing points help a trader understand the market's direction and where meaningful shifts are occurring. The indicator does not generate buy or sell orders on its own. Instead, it highlights the levels where market structure changes, and the trader can incorporate that information into their broader trading approach.

A trader can choose to use either intermediate swing points or long-term swing points as the foundation for their decision-making. The idea is simple. When an intermediate-term swing low forms, it suggests that buyers have stepped in to create a temporary floor in prices. A trader who prefers to follow the trend may choose to take a long position at this point and hold it until an intermediate swing high invalidates the structure. The same logic applies in reverse. If an intermediate-term swing high forms, this may signal that sellers have created a ceiling. A trader can then consider entering a short position and remain in that trade until a new intermediate swing low appears.

Long-term swing points can be used in the same way, but they provide signals that appear less frequently and reflect broader shifts in market structure. This makes them more suitable for traders who prefer a slower trading style or those who work on higher timeframes. A long-term swing low may act as a strong signal for a potential longer bullish phase, while a long-term swing high may warn of an extended bearish move. These long-term structures help traders hold positions with greater confidence, especially when combined with trend filters or higher-timeframe analysis.

The indicator can also support trade management decisions, especially regarding exits and stop-loss placement. For example, if a trader is in an extended position and the price retraces below the previous intermediate swing low, this may indicate that the structure has broken. In such a case, exiting the trade becomes a logical choice. Similarly, a short position can be closed if the price climbs above the previous intermediate swing high. This use of swing points for managing ongoing trades helps traders protect profits and avoid holding positions when the market has clearly shifted against them.

While this indicator gives valuable structure-based information, it should always be used as a complement to a complete trading plan. Swing points alone are not enough for consistent decision-making. Traders are encouraged to combine them with other forms of confluence, such as trend direction, liquidity zones, moving averages, or volume analysis. The goal is to build a robust process where the swing structure helps confirm what the trader already sees in the market.

In summary, this indicator provides a clear and systematic way to interpret market structure. It highlights momentum shifts, helps identify trend continuation and reversal points, and offers a straightforward approach to entries and exits. When used correctly and combined with other reliable tools, it can significantly improve a trader's ability to read the market and make informed decisions.

Conclusion

This article shows how to turn the market structure concepts introduced by Larry Williams in his book, *Long-Term Secrets to Short-Term Trading,* into a fully functional MQL5 indicator. Step by step, we designed the plots, prepared the buffers, and built the algorithms that extract short-term and intermediate-term swing points directly from price data. The final result is a practical tool that traders can place on their charts to visualize market structure with clarity.

Beyond the indicator itself, the reader has gained hands-on experience with core MQL5 concepts, including buffer management, plot configuration, and structured indicator logic. These skills can serve as a starting point for building long-term swing detection, alerts, automated strategies, or any other enhancements they may want to add. This is a real project with a real application, and it should serve as both a learning resource and a foundation for future development.