

# MIM Query Language Overview



12301 Research Blvd.  
Building IV, Suite 410  
Austin, TX 78759

U.S. Help Desk Phone: +1-800-546-9646 (or direct +1-512-697-3000), select ext. 3400

U.K. Help Desk Free Phone: 0800 032 6063

Europe Help Desk Phone: +44 20 7190 2947

Help Desk Email: [support@lim.com](mailto:support@lim.com)

+1-512-697-3001 (Fax)

Part Number: 089\_8

Date: September 18, 2008

Copyright © 2000-2008 by Logical Information Machines, Inc.

Patented May, 1995 U.S. Patent No. 08/392, 612

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Logical Information Machines, Inc.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013.

Logical Information Machines, Inc.  
120 North LaSalle Street  
Suite 2150  
Chicago, IL 60602

(312) 456-3000

Product names mentioned herein are for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

While every precaution has been taken in the preparation of this manual, we assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein. Logical Information Machines, Inc. may revise this publication from time to time without notice.

---

# Table of Contents

<b>MIM Query Language Overview</b> .....	<b>1</b>
Introduction .....	1
Attributes .....	1
<attribute><time offset> .....	2
<attribute unit><attribute> .....	3
Variables as Attributes .....	3
Conditions .....	4
Comparing One Attribute To Another .....	4
Comparing a Date To a Pattern .....	5
Comparing a Time To a Pattern .....	8
If Then Else Statements .....	8
Repeating Conditions .....	8
Repeat Times .....	8
Query Blocks .....	8
Overview .....	9
DO-SHOW-WHEN .....	9
ORDER .....	10
DO EVERY PERIOD .....	12
DO EVERY TRADE ON ENTRY/EXIT .....	13
Multiple Query Blocks .....	14
Initialization Statements .....	14
FINALIZE Added .....	14
FINAL SHOW Added .....	15
Query Language Components .....	15
The Major and Minor Query Blocks .....	15
Query Execution Model .....	16
<b>Index</b> .....	<b>17</b>



---

# MIM Query Language Overview

## Introduction

The MIM query language (MQL) has been designed for building models of time series data for forecasting and trade simulation. To facilitate these tasks LIM has included many of the common terms and calculations needed for technical and fundamental analysis.

The query is created, and once executed, evaluation proceeds through time, evaluating each query block in the sequence found in the query for a given date before changing the date, and evaluating the blocks again.

Query blocks typically contain one or more conditions that define an event, and an action to perform when the event is found. Actions include showing the data requested, placing a simulated trade order, or evaluating a DO block.

Before we can start building queries, we must first understand how attributes and conditions are used.

## Attributes

Numbers used in a MIM query can be derived in two ways. As a simple time series which has been retrieved unaltered from the database or as a computed time series. Computed time series are created using arithmetic combinations of simple time series, functions, and constants. Both types are handled the same by the MIM, and are termed attributes.

An example of a simple time series is “Close of DJIA”. This data is retrieved from the MIM database unaltered, and will yield a single number for each date in the time series, which is the closing price of the Dow Jones Industrial Average. When shown in a report, the attribute will appear as a list of numbers, next to the appropriate date like this.

01/04/2000	10997.93
01/05/2000	11122.65
01/06/2000	11253.26
01/07/2000	11522.56
01/10/2000	11572.20

An example of a computed time series is “Close of DJIA / Close of SPX”. This attribute will yield a single number for each date both DJIA and SPX were active. Note that since this computed time series is composed of two attributes, it is implied that it could have been composed of two more complex attributes, such as the

following, “(Close of DJIA / Close of SPX) \* 100”. Here we are using an attribute derived from a constant (100), as well as the obvious simple attributes.

Additional simple attributes include:

- High of US
- Volume of DELL
- DistToEarth of MOON
- High of IBM – Low of IBM
- Cash of GC – Close of GC

Additional computed attributes include:

- 10 day average of CL
- 3 week move of JY
- 4 day highest of High of US
- simple\_rsi ( High of US, 9 days )
- IF 1 day move of US is more than 0 THEN 1 day move of US ENDIF
- entry\_price
- bollinger\_high ( S, 10 days, 1 )
- periods\_in\_range from today to next expiration day of LH



Where <attribute> is used, an attribute derived by any means can be substituted.

There are two other important methods of manipulating attributes, which are relevant to the fact that we are looking at time series data. Those are time offset, and time units, known here as attribute units.

## <attribute><time offset>

Time offset refers to some point relative to the current observation time. Some examples are 1 day ago and 1 day later. These statements can be used to modify an attribute, yielding a statement such as “Close of DJIA 1 day ago”, or “Close of SPX 1 day later”. These time offsets are denoted <time offset>. So the general statement is <attribute> <time offset>. This combination of attribute and time offset resolves to an attribute. This capability allows the statement (Close of DJIA – Close of DJIA 1 day ago), returning the difference in the closing price of DJIA from yesterday to today.

Some examples of time offset attributes:

- High of US 1 day ago
- Volume of DELL 1 week ago
- DistToEarth of MOON 1 month later
- High of IBM 1 unit ago – Low of IBM 1 unit ago
- Cash of GC 3 days ago – Close of GC 3 days ago

## <attribute unit><attribute>

Attribute units refer to the time granularity of the attribute in question. Some examples are hourly, daily, weekly, monthly. This syntax yields statements such as “The hourly Close of DJIA”, and “The Weekly High of DJIA”. When used in complex attributes we see statements such as (The weekly High of DJIA – The Daily Close of DJIA). For further explanation of attribute units syntax see the [XMIM User Guide](#).

Attributes with modified attribute units:

- 10 minute High of US
- 1 hour High of US
- The daily High of US or 1 day High of US
- The Weekly Volume of DELL or 1 week Volume of DELL
- The Monthly High of IBM or 1 month High of IBM

The fully modified attribute can be generalized to yield the following form <attribute units> <attribute> <time offset>.

Complex attributes:

- (The daily Close of US – The weekly Low of US 1 week ago) / (The weekly High of US 1 week ago – The weekly Low of US 1 week ago)

Which indicates where today’s close is in comparison to last weeks range.

- 10 minute Close of US – The Daily 10 day average of Close of US

Which compares the 10 minute close to the 10 day average of the daily bars.

## Variables as Attributes

Variables as attributes deserve a special note. Variables can be assigned to and evaluated with the MIM language for use as counters and flags. Assignment will be handled in the section on conditions, but it should be noted here that while a variable can be manipulated like an attribute there are some special considerations to keep in mind. Primary is the fact that while you can offset a variable in time (mvVariable 1 day ago) the value returned will be whatever the value is currently. To understand this, assume that the value is stored in a single memory register, or one-dimensional array. No matter how the value is offset, it will always be the value that is currently in memory.

## Repeating Attributes

### Previous-Current-Next PeriodP

- Repeated for the [previous n <condition>] to [next n <condition>]
- Repeated for the previous 10 to next 10 days
- Repeated for the previous 1 and current day
- Repeated for the current and next 11 days

### Period

- Repeated for <condition>
- Repeated for 1 week
- Repeated for the entire week
- Repeated for 1 == 1

### Repeat From-To

- Repeated from <time offset> to <time offset>

### Repeat For...To...By...

- Repeated for <variable> from w to x by y

### In Date/Time Condition

- Repeated <xx><date condition | time condition>
- Xx = in, by, before, after, since
- Subset of conditions = date & time conditions,

## Conditions

Conditions can be constructed in four ways:

1. Comparing one attribute to another.
2. Comparing a date to a pattern.
3. Comparing a time to a pattern
4. A condition macro.

## Comparing One Attribute To Another

Comparing one attribute to another is the most obvious condition available. The comparison is made using one of the seven following conditional operators. Each operator has logical modifiers associated with it. The

modifiers “is” or “is not”, can be used with operator 1-5, and the modifiers “does” or “does not”, are for use with operators 6 and 7. If there is shorthand for the statement, it will follow in parentheses.

1. more than (>)
2. at least (>=)
3. less than (<)
4. at most (<=)
5. exactly (==)
6. crosses above
7. crosses below

The general form for this type condition is <attribute> <conditional operator> <attribute>.

Examples of comparing attributes:

- Close of US is more than Close of US 1 day ago
- Close of US is not more than Close of US 1 day ago
- Close of US is exactly High of US
- Low of US >= 1 week High of US 1 week ago
- Close of IBM crosses above 50 day average of IBM
- Close of IBM does not cross above 50 day average of IBM

Note the difference of these two statements:

- Close of IBM is more than 50 day average of IBM
- Close of IBM crosses above 50 day average of IBM

The point to be made here is that when “crosses above” is used, there is a single day when the condition is true. The day when the close is above the average, but it was not the day before. This would be useful in evaluating the effect of IBM crossing above its average by showing using the following attribute.

- move from today to 1 week later of close of IBM

By using the “crosses above” statement, there will be 1 number returned for each time the close crosses above the average. If we had used the “is more than” operator, there would be a number showed each day IBM closed above its average, so that the event being studied would no longer be “when IBM closed above its average”.

## Comparing a Date To a Pattern

A date comparison always begins with the statement Date is, or Date is not and is followed by the pattern to match.

The following patterns are available.

- Date is mm/dd/yyyy

Any of the 3 tokens can be either a number, a variable, or the wild card “\_”, (which is always true).

Examples:

- Date is 8/2/1999
- Date is 8/myVar/1999
- Date is 8/2/\_ (matches Aug. 2nd every year)

- Date is date:DateVar

DateVar contains the Julian date to be matched. The Julian date can be calculated using the construct date function as follows.. DateVar = construct\_date( 8, 2, 1999 ). Then the date comparison Date is date:DateVar can be made.

- Date is date:myDateVar

- Date is from <date> to <date>

Where <date> is any valid date created using the patterns outlined in this section.

- Date is from 1/1/1999 to 3/1/1999
- Date is from 8/2/\_ to date:dateVar

- Date is <calendar name>

Where <calendar name> is any weekday name (Monday to Sunday), month name (January to December), quarter (1st to 4th), and year.

- Date is Monday
- Date is January
- Date is 3rd Quarter
- Date is 1999

- Date is <special day>

Where <special day> are date patterns that are pre-loaded into the MIM.

- Date is New Years Day
- Date is Labor Day

- Date is in <date file>

Where date file is the path to any ASCII file containing dates of the format yyyyymmdd or mm/dd/yy. LIM updates 2900+ date files each day which describe events from treasury auctions to phases of the moon.

These files can be found at /home/lim/dates.

- Date is in “/home/lim/dates/user/opc\_meet.date”
- Date is in “/home/lim/dates/user/m\_supply.rpt”

- Date is <symbol specific date>

Where <symbol specific date> is a combination of a symbol, and an event name. The valid event names are Expiration, First data, Last data, Ex dividend, Release, and Split. The event name may be followed by the word “day” to improve readability, and case is optional.

- Date is US Expiration
- Date is US expiration day
- Date is US first data day
- Date is <time period> <boundary>

Where <time period> is a standard time period and boundary is either open or close.

- Date is weekly open
- Date is day open
- Date is nth [to last] <small period> of <big period>

Where nth is the number of small periods to count from the big period boundary. <small period> is the day, week, or any day name from Monday to Sunday. <big period> is week, month, quarter, year, or any month name from January to December. This (for example) will count the number of days to some larger time boundary, and use of the optional “to last” syntax reverses the direction of counting.

- Date is 1st day of the week
- Date is 3rd to last week of the year
- Date is within n <time period>

Where time period is a standard time period.

- Date is within 1 week
- Date is within 9 months

This is perhaps not obvious after all the definitions that have gone before. The offset is relative to the actual date at the time the query is run, such that since I am writing this on January 8, 2000, when I use the condition Date is within 1 week, then the results will be different (by one day) than if I ran the query 1 day later.

## Comparing a Time To a Pattern

A time comparison always begins with the statement 'Time is, or 'Time is not and is followed by the pattern to match. The following patterns are available.

- Time is hh[:mm [am | pm]
- Time is from <time> to <time>
- Time is <time period> <boundary>
- Time is [first|last] N <time period> of trading
- Time is time:timeVar
- Time is <system variable>
- Time is <symbol specific time>

## If Then Else Statements

IF-THEN-ELSE statements use the general form:

```
IF <condition>
THEN <condition>
ELSE <condition>
ENDIF
```

Where <condition> is any legal condition outlined within this chapter.

## Repeating Conditions

- <Repeat times><repeat attribute>

## Repeat Times

- { <condition> } <repeat times> <repeat attribute>
- Nth [to last] time in
- Always | once
- Repeated n [successive | percent of | % of] time

## Query Blocks

The following section describes how query blocks are constructed.

## Overview

Query blocks form the skeleton of the MIM query language and contain two functional sections, the event or scenario, and the action. The event is the combination of one or more conditions which triggers the associated action. The actions fall into three general categories:

- Show a report of attribute values
- Simulate trade placement and position management
- Process variable assignments

In all cases, the event conditions are evaluated in a “fall through” manner. If a statement is found to be true, evaluation proceeds to the next statement. If a statement is false, evaluation of the event stops, its associated action is not triggered, and execution proceeds to the next query block, or next data point. In order for the action to be triggered, all statements must evaluate as true.

The following sections detail each query block type by outlining the processing sequence and the nature of the associated action. Examples of each type are provided.

## DO-SHOW-WHEN

FORM	ORDER of EXECUTION	ACTION
DO <conditions>	1: WHEN	SHOW – reports attribute values and optionally calculates summary statistics about the columns of data reported.
SHOW <attributes>	2: DO	
WHEN <conditions>	3: SHOW is	

This models the event in the WHEN block, then performs calculations and flag settings in the DO, and reports attribute values with the SHOW.

Example 1:

```
DO
    myCounter = myCounter + 1
SHOW
    1: myCounter
    2: percent_move from today to 2 days later of Close of DJIA
WHEN
    Close of DJIA crosses above 52 week average of Close of DJIA 1 day ago
```

English: This query answers the question, ‘How many times has the DJIA crossed above its 52 week average, and after each crossing, what was the percent change over the next 2 days?’



In all 3 sections, zero or more statements could have been used. When a section has zero statements, the related keyword can be left off as in example 2.



In the second show attribute, we have legally substituted the percent sign for the word percent.

Example 2:

```
SHOW
  1: move from today to 2 days later  of Close of DJIA
WHEN
  Date is after 1998
AND
  Date is Wednesday
```

English: This query answers the question, ‘How much has the DJIA moved in two days since its closing when the date is a Wednesday after the year 1998’.



The DO section is not being used and the keyword was omitted.

The following options affect the DO-SHOW-WHEN block: See the [XMIM User Guide](#) for details on these options.

- Execution units
- Attribute Units
- Execute Options

## ORDER

FORM	ORDER of EXECUTION	ACTION
<label>: ORDER	1: WHEN	ORDER – order placed when condition is true. EXIT – position is exited when condition is true.
<label>: <order placement rules>	2: <order placement rules>	
WHEN <conditions>	3: EXIT	
EXIT <conditions>		

The ORDER block can best be understood by looking at how a trade is made in the real world.

First an event is observed. This is represented by the WHEN condition. Next an order is placed (not to be confused with executing a trade). Generally the order contains rules of execution, either implied, such as wait

until the market opens tomorrow, or specified such as only if we trade down to a specific price over the next 3 days. This is handled in the <order placement rules>. If the <order placement rules> are satisfied, the trade is filled, if they are not the order is canceled. Finally, the EXIT block is evaluated each day there is a position on, and when true, the position is closed.

```
1: ORDER
  1.1: Buy 1 contract of JY
WHEN
  Close of JY crosses above 30 day average of JY
EXIT
  Close of JY crosses below 10 day average of JY
```

English: This ORDER block will get long 1 contract of JY when JY closes above its 30 day average, and exit when JY closes below its 10 day average.



In the DO-SHOW-WHEN block, some parts can be omitted, namely the WHEN, and EXIT sections. When these sections have no conditions, the keywords can be omitted.



The text '1:' and '1.1' are labels and can be replaced with text strings as long as they do not contain white spaces or special characters.

The following options affect the ORDER block:

- Execution Units
- Attribute Units
- Execute Options
- Profit-Loss Options

## ORDER Placement Rules

The following shows the order placement rules:

```
<trade><time offset><size> <instrument> <entry rules> <exit rules>
```

where:

- <trade> - This is the type trade to make, either Buy or Sell
- <size> - This is the quantity to trade, it can be an <attribute> shares; percent; dollars
- <instrument> - This is the symbol to trade, it can also be an <attribute>
- <entry rules>–

```
[Enter [on the (open | close | market)]
  [ with [entry_stop at <attribute>]
    [limit at <attribute>] ] ]
```

- <exit rules> –

```
[Exit [on the (Open | Close| Market)]
    [ with
        [stop_loss at <attribute>]
        [trail stop_loss at <attribute>]
        [take profit at <attribute>]
        [trail profit at <attribute>]
    ]
]
```

Example:

```
1: ORDER
  1.1: Buy 1 contract of US
      Enter on the market
      Exit with stop_loss at Low of US 1 day ago
            trail stop_loss at Low of US 1 day ago
            take profit at Close of US 1 day ago + 10 day average of range of US
            trail profit at Low of US 1 day ago
```

## DO EVERY PERIOD

The following query example of DO EVERY PERIOD generates an analog chart. The OLD symbol is the historical one, the @ is the instrument you are interested in. In this example they are the same but it does not always have to be the same (for instance you could compare the current stock market against soybeans). The CorTimePeriod is the look-back window for the correlated history search. If CorTimePeriod is set to 100, for example, it looks back at the last 100 days of history in the @ ticker and searches the Old ticker for 100-day periods whose coefficient of correlation is strongest, meaning that the movements are closest to the current periods. In the resulting chart, the correlation value is plotted in the lower pane.

```
%graph.plot.colors: "black blue red green yellow cyan blue magenta orange pink"
%graph.back.color: "white"
%graph.scalewith: "@" "@"
%graph.scalewith: "Old" "Old"
```

```
LET
  Old = SPX
  @ = SPX
  ATTR CorTimePeriod = 100
  ATTR StartYear = 1950

INITIALIZE
  daysAgo = 0
  AND
  beginDate = construct_date ( 1, 1, StartYear )
  AND
  BestValue = 0
  AND
  BestDate = 0

LET
  ATTR iteration = 1, 2

DO EVERY 1 day
  iteration is exactly 1
```

```

AND
  Old CorTimePeriod days ago is DEFINED
AND
  Date is after 1 day before date:beginDate
AND
  Date is before CorTimePeriod days before Old last_data_day
AND
  thatMany = periods_in_range from today to Old last_data_day
AND
  thisCor = correlation ( Old, @ thatMany days later, CorTimePeriod days
  )
AND
  IF
    thisCor is more than BestValue
  THEN
    BestDate = current date
  AND
    BestValue = thisCor
  ENDIF

SHOW
{
  Old: BAR of Old
  @: BAR of @ thatMany days later
  Correlation: IF
    Old thatMany days later is DEFINED
  THEN correlation ( Old, @ thatMany days later,
    CorTimePeriod days )
  ENDIF
}
  repeated from CorTimePeriod units ago to CorTimePeriod units later
WHEN
  iteration is exactly 2
AND
  Date is after 1 day before date:beginDate
AND
  Date is date:BestDate
AND
  thatMany = periods_in_range from today to Old last_data_day

```

## DO EVERY TRADE ON ENTRY/EXIT

The DO ON ENTRY block will be executed each time a trade entry occurs.

```

DO ON ENTRY
someVariable = 1

```

The DO ON EXIT block will be executed each time a trade exit occurs.

```

DO ON EXIT
someVariable = 0

```

Options which effect the DO ON ENTRY / EXIT block:

- Execution Units
- Attribute Units
- Execute Options

## Multiple Query Blocks

A single query can have multiple query blocks. This feature can be used to debug a query by showing the values and variables. It can also be used to clarify meaning by separating variable initialization from general manipulation. Profit and loss statistics can be displayed in this manner as well. Generally the blocks are executed in the order they appear in the query. Any query block “knows” what is going on in another with the exception of some special cases related to ORDER blocks. This allows the passing of variables from one query block to another.

Lexical scoping is a measure pertaining to ORDER blocks.

An example of this is the system variable entry\_price. This variable can be referred to safely within the ORDER block to which it refers. Once execution moves beyond that particular ORDER block, the value becomes nonsensical in that it is unknown what ORDER block is referred to.

## Initialization Statements

Initialization statements are used to initialize (zero), variables used in a query.

```
INITIALIZE
numberTrades = 0
AND
endingEquity = 0
```

The INITIALIZE block will be executed once per query execution, depending on where the block is located. When this block is placed above the LET, the variables initialized will survive all LET iterations.

## FINALIZE Added

The FINALIZE block will be executed once per query execution, depending on where the block is located.

```
FINALIZE
myCommission = numberTrades * 25
```

## FINAL SHOW Added

The FINAL SHOW block will be executed once per query execution, depending on where the block is located.

```
FINAL SHOW
  1: myCommission
  2: numberTrades
  3: totalEquity
  4: numberDays
```

## Query Language Components

The query language contains three major query blocks, which must appear in the following order:

1. Execute Options
2. Initialization Statements
3. Minor Query Blocks

Each major query block contains one or more minor query blocks. The ordering of the minor query blocks within the major query block is flexible, depending on the desired outcome.

## The Major and Minor Query Blocks

1. Execute Options
  - Execution time period selection
  - Graphics configuration options
  - Profit and Loss testing options
2. Initialization Statements
  - LET statements
  - INITIALIZE statements
  - FINALIZE statements
  - FINAL SHOW action
  - TOP BOTTOM AS-LONG-AS report filtering
3. Query Blocks
  - DO SHOW WHEN
  - DO EVERY UNIT
  - DO EVERY TRADE
  - ORDER WHEN EXIT

## Query Execution Model

The five steps of a query sequence are:

1. Execution option implementation
2. LET and INITIALIZATION statement evaluation
3. Minor query block evaluation
4. FINAL and FINAL SHOW statement evaluation
5. Report filtering

However, given that LET statements may assign a range of values to a variable, a better picture of the execution sequence can be illustrated with the following code.

```
Implement execution options
Evaluate the INITIALIZE statements
For each LET value.
    For each date to test
        For each query block
            IF the condition is true
            THEN perform the action
        End For
    End For
End For
End For
Evaluate the FINAL statements
Implement TOP / BOTTOM / AS_LONG_AS filters
```



There can be n LET statements within the second major query block, therefore for each one, a “For each LET” loop is added to the nest.

Time Offset – syntax detail

- 1 day ago / later
- previous / next <condition>

Attribute Units – syntax detail

- [1-99] [minutes | hours | days | weeks | months | quarters | years]

# Index

## A

Attribute Unit, 3

Attributes, 1

## C

Comparing a Date To a Pattern, 5

Comparing a Time To a Pattern, 8

Comparing One Attribute To Another, 4

Conditions, 4

## D

DO EVERY PERIOD, 12

DO EVERY TRADE ON ENTRY/EXIT, 13

DO-SHOW-WHEN, 9

## F

FINAL SHOW, 15

Finalize, 14

## I

If Then Else Statements, 8

In Date/Time Condition, 4

Initialization Statements, 14

## M

Multiple Query Blocks, 14

## O

ORDER, 10

ORDER Placement Rules, 11

## P

Period, 4

Previous-Current-Next Period, 4

## Q

Query Blocks, 8

Query Execution Model, 16

Query Language Components, 15

## R

Repeat For...To...By..., 4

Repeat From-To, 4

Repeat Times, 8

Repeating Attributes, 4

Repeating Conditions, 8

## T

time offset, 2

## V

Variables, 3

