Micro-line Canada Ltd.

Application Program Interface

Market By Price Managed Service

Version 1.0

June 6, 2006

1. Notice

This specification is being provided strictly for information purposes, to assist in the development of systems to interact with Micro-line Canada Ltd. Market By Price Managed Service.

2. Revision History

Version	Date	Author	Description
1.0	June 6, 2006	SO – Micro-line	Initial Version

TABLE OF CONTENTS

3. Preface

1.1 Scope

This guide describes the Application Program Interface to the Market By Price Managed Service.

1.2 Audience

This guide is for developers building a programmatic interface to the Market By Price Managed Service.

1.3 Typographical conventions

- Source code appears in `mono-spaced text` with a gray background. For example:

```
#/bin/sh
echo "started"
date
echo "finished"
```

- Important information or changes from previous versions of the document are highlighted.
- Directory names and filenames usually appear in a ***bold italic*** font. For example:

  etc/populate_parameters_nexgen-primary.sql

- Commands typed by the user appear in `mono-spaced` font. For example:

  make upgrade

- Cross-references to other guides appear in regular italic enclosed in quotation marks. For example:

  See the "*Administration – Market By Price Managed Service*" document.

1.4 Definitions, Acronyms, and Abbreviations

API

The interface that a computer system, library or application provides in order to allow requests for service to be made of it by other computer programs, and/or to allow data to be exchanged between them (from http://wikipedia.org/).

Berkeley Socket

The **Berkeley sockets** application programming interface (API) comprises a library for developing applications in the C programming language that perform inter-process communication, most commonly across a computer network.

Berkeley sockets (also known as the BSD socket API) originated with the 4.2BSD Unix operating system (released in 1983) as an API. Only in 1989, however, could UC Berkeley release versions of its operating system and networking library free from the licensing constraints of AT&T's copyright-protected Unix.

The Berkeley socket API forms the *de facto* standard abstraction for network sockets. Most other programing languages use a similar interface as the C API. (from http://wikipedia.org/).

BoardLot

A standard trading unit as defined in UMIR (Universal Market Integrity Rules). The board lot size of a security on Toronto Stock Exchange or TSX Venture Exchange depends on the trading price of the security, as follows:

- Trading price per unit is less than $0.10 - board lot size is 1,000 units
- Trading price per unit is $0.10 to $0.99 - board lot size is 500 units
- Trading price per unit is $1.00 or more - board lot size is 100 units

CL2

TSXV level 2 orders and trades data feed sourced by TSX Datalinx.

COM

Component Object Model is a Microsoft platform for software componentry introduced by Microsoft in 1993. It is used to enable interprocess communication and dynamic object creation in any programming language that supports the technology. The term COM is often used in the software development world as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies. Although COM was introduced in 1993, Microsoft did not begin emphasizing the name COM until 1997.

Although it has been implemented on several platforms, it is primarily used with Microsoft Windows. COM is expected to be replaced to at least some extent by the Microsoft .NET framework (from http://wikipedia.org).

DDE

Dynamic Data Exchange is a technology for communication between multiple applications under Microsoft Windows and also OS/2. Although still supported in even latest Windows versions, it has mostly been replaced by its much more powerful successors OLE, COM, and OLE Automation. However, it is still used

in several places inside Windows, e.g. for Shell file associations (from http://wikipedia.org).

DLL

Dynamic-link library, also referred to as **dynamic link library** (without the hyphen), is Microsoft's implementation of the shared library concept in the Microsoft Windows operating systems. These libraries usually have the file extension DLL, OCX (for libraries containing ActiveX controls), or DRV (for legacy system drivers) (from http://wikipedia.org)..

DOS Shell

cmd.exe is the command line interpreter on OS/2 and on Windows NT-based systems (including Windows 2000, XP, and Server 2003). It is the equivalent of command.com in MS-DOS and Windows 9x systems, or of the shells used on Unix systems from (http://wikipedia.org).

Excel

Microsoft Excel is a spreadsheet program written and distributed by Microsoft for computers using the Microsoft Windows operating system and for Apple Macintosh computers. It features an intuitive interface and capable calculation and graphing tools which, along with aggressive marketing, have made Excel one of the most popular microcomputer applications to date. It is overwhelmingly the dominant spreadsheet application available for these platforms and has been so since version 5 in 1993 and its bundling as part of Microsoft Office (from http://wikipedia.org).

FAQ

Frequently Ask Questions - The term refers to listed questions and answers, all supposed to be frequently asked in some context, and pertaining to a particular topic (from http://wikipedia.org).

MBP

A tabular display of the number of orders, aggregated by price, at each price level for a given symbol and market (Boardlot, Oddlot or Terms).

Oddlot

A number of shares that are less than a board lot, which is the regular trading unit decided upon by the particular stock exchange. An odd lot is also an amount that is less than the par value of one trading unit on the over-the-counter market. For example, if a board lot is 100 shares, an odd lot would be 99 or fewer shares.

OS

An operating system is an essential software program that manages the hardware and software resources of a computer. The OS performs basic tasks, such as controlling and allocating memory, prioritizing the processing of instructions, controlling input and output devices, facilitating networking and managing files (from http://wikipedia.org).

RTD

Microsoft Excel 2002 provides a new worksheet function, RealTimeData, which allows you to call a COM Automation server for the purpose of retrieving data in real time.

When you need to create a workbook that includes data that is updated in real time -- for example, financial or scientific data -- you can now use the RTD worksheet function. In earlier versions of Excel, DDE is used for that purpose. However, the RTD function, which is based on COM technology, provides advantages in terms of robustness, reliability, and convenience. RTD depends on the availability of an RTD server to make the real-time data available to Excel.

The RTD function retrieves data from an RTD server for use in the workbook. The function result is updated whenever new data becomes available from the server and the workbook is able to accept it. The server waits until Excel is idle before updating, which relieves the developer of having to determine whether Excel is available to accept updates. The RTD function differs from other functions in this regard, because other functions are updated only when the worksheet is recalculated.

(Excerpted from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexcl2k2/html/odc_xlrtdfaq.asp)

TCP

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet_protocol_suite. Using TCP, applications on networked hosts can create connections to one another, over which they can exchange data or packets. The protocol guarantees reliable and in-order delivery of sender to receiver data. TCP also distinguishes data for multiple, concurrent applications (e.g. Web server and e-mail server) running on the same host (from http://wikipedia.org).

Terms

Orders which must trade under special conditions. For example, a cash order will be settled sooner than the usual three-day settlement period.

TL2

TSX level 2 orders and trades data feed sourced by TSX Datalinx.

TSX

The Toronto Stock Exchange.

TSXV

The TSX Venture Exchange.

3.1 References

- Toronto Level 2 / TSX Venture Exchange Level 2, *Functional Specifications*, TSX Markets, September 2005
- STAMP Specification, Version 4.0, TSX Markets, January 2003
- Administration - Feed Reader Managed Service, Micro-line Ltd., Version 1.0
- Application Program Interface – Feed Reader Managed Service, Micro-line Ltd., Version 1.0
- Administration – Market By Price Managed Service, Micro-line Ltd., Version 1.0

1.5 Wikipedia License

Text copied from Wikipedia is made available under the following license:

http://en.wikipedia.org/wiki/Wikipedia:Text_of_the_GNU_Free_Documentation_License

4. Product Description

The Market By Price Managed Service API provides:

- A TCP socket interface to the Toronto Stock Exchange (TSX/TSXV) Market By Price (MBP) (Boardlot, Oddlot and Terms) order books, delivering a sequenced, reliable stream of data in field and record delimited format. This interface is operating system (OS) and compiler agnostic, and allows applications to be developed for any operating system and compiler that supports the establishment of TCP connections and sessions.
- A Microsoft Excel Real Time Data (RTD) interface to the Toronto Stock Exchange (TSX/TSXV) MBP (Boardlot, Oddlot and Terms) order books. This interface is limited to the Microsoft OS and the Excel application. This interface is inherently unreliable due to the design of the Microsoft RTD interface as described below.

The Market By Price service depends on the Micro-line Feed Reader Managed Service to provide the CL2 and TL2 data used to maintain the order books.

See the "*Administration – Feed Reader Managed Service*" and "*Application Program Interface – Feed Reader Managed Service*" documents for more details.

1.6 TCP Socket Interface

1.6.1 Overview

The Micro-line Market By Price Service provides a TCP socket interface to the MBP order book for each symbol on the TSX/TSXV exchanges for the Boardlot, Oddlot and Terms markets. The entire depth of the order book is maintained and disseminated to client applications in real time. Client applications make a TCP connection to the Market By Price Service IP address and port to receive a sequenced reliable stream of TSX MBP data for the specified symbols and markets. The data is field and record delimited for easy manipulation by client applications.

Refer to the *"Administration – Market By Price Managed Service"* document for the location of the Market By Price Service, referred to as "**<MLMBP_01_IP>**" , "**<MLMBP_02_IP>", ..., "<MLMBP_NN_IP>**" below. The Market By Price Service processing modules are assigned IP addresses on the client LAN, numbered from **01** to **NN**, where **NN** is the number of Market By Price Service processing modules installed, usually **02**.

The Market By Price Service accepts client TCP connections on port **9900**.

An MBP client connects to the MBP service using the one of the specified IP address and the specified port number. It sends one or more messages to the MBP service specifying which symbol(s) and markets it is interested in. For each specified symbol and market it receives a stream of MBP messages.

Each message sent by a client is newline (ASCII decimal 10, hex 0A) record delimited and space (ASCII decimal 32, hex 20) field delimited.

Each MBP message sent to a client is newline (ASCII decimal 10, hex 0A) delimited.

Each MBP message consists of 6 "**~**" (ASCII decimal 126, hex 7E) delimited fields.

An example of the stream of messages passing between a client interested in the ALE Boardlot MBP is:

Client to MBP	***MBP to Client***	***Commen***
*<NL>		Interactiv e session
start ALE Boardlot<NL>		Start updates for ALE Boardlot MBP
	ALE~B~S~U~0~ OK<NL>	Request to start ALE Boardlot OK
	ALE~B~Q~U~0~ Symbol<NL>	First full quote field name
	ALE~B~Q~U~1~ ALE<NL>	First full quote field value
	...	Remainin

Client to MBP	*MBP to Client*	*Commen*
		g full quote field name/value pairs
	ALE~B~B~U~0~3 |16.000|4500| <NL>	First initial buy side price level
	...	Remaining initial buy side price levels
	ALE~B~A~U~0~2 |16.030|1200| <NL>	First initial ask side price level
	...	Remaining initial ask side price levels
	ALE~B~A~U~35~ 1|20.900|300| <NL>	Final initial ask side price level
	...	Asynchronous updates as ALE Boardlot MBP changes.

In the above example, the client connects to the MBP service, sends initial commands specifying interactive service and starts updates for symbol **ALE** market **Boardlot**. The MBP service sends back a status response indicating that the request is valid and then sends the initial order book as multiple messages.

The the MBP service asynchronously send update messages as the state of the MBP order book is updated in real time.

The MBP service supports 2 types of sessions:

1. Single symbol/market session for clients interested in a single symbol/market. For example, the reference Java MBP client uses this session type.
2. Interactive session for clients interested in multiple symbol/market combinations. This session type provides the ability to start and stop symbol/market updates dynamically.

4.1 Single Session

1.1.1 Client to MBP Message

TBD

1.1.2 MBP to Client Messages

TBD

1.2 Interactive Session

1.2.1 Client to MBP Messages

After connection to the MBP service, the first message the client must send is:

```
*<NL>
```

This informs the MBP service that this an interactive session. There is no response from the MBP service.

The client can then send messages of the form:

```
<command> <symbol> <market><NL>
```

The message consist of 3 space (ASCII decimal 32, hex 20) delimited fields:

- **<command>** - one of the values "**start**" to start receiving updates for the specified **<symbol>** and **<market>** or "**stop**" to stop receiving updates for the specified **<symbol>** and **<market>.**
- **<symbol>** - a valid TSX or TSXV symbol in TSX ticker format.
- **<market>** - one of "**Boardlot**", "**Oddlot**" or "**Terms**" for the Boardlot, Oddlot or Terms markets respectively.

The MBP service will respond with a **status** message, described below.

1.2.2 MBP to Client Messages

MBP to Client messages consist of 6 "**~**" (ASCII decimal 126, hex 7E) delimited fields:

Index	*Name*	*Description*
1	Symbol	A symbol specified in a previous Client to MBP **start** messsage.
2	Market	A market specified in a previous Client to MBP **start** messsage, "**B**" for **Boardlot**, "**O**" for **Oddlot** or "**T**" for T**erms**.
3	Side	The side of the market that the message pertains to, "**Q**" for Full Quote, "**B**" for Buy/Bid or "**A**" for Ask/Sell.
4	Operation	The operation to perform on the specified **Side**, **Offset** and **Value**; "**U**" to update the **Value** at the specified **Offset**, "**I**" to insert a new **Value** after the specified **Offset** and "**D**" to delete the specified

Index	*Name*	*Description*
		Offset.
5	Offset	An offset into the market **Side**, first offset is 0(zero).
6	Value	The value to apply using the specified **Side**, **Operation** and **Offset**.

1.2.2.1 Status Message

This message is sent in response to a **start** or **stop** message sent by the client.

If the client sends the message:

```
start NT Boardlot
```

A typical response is:

```
NT~B~S~U~0~OK
```

The client will now receive asynchronous messages for symbol **NT** market **Boardlot**.

The 6 fields of the message are:

1. Symbol
2. Market
3. "**S**" to indicate that this is a status message.
4. Operation is not applicable and always set to "**U**".
5. Offset is not applicable and is always set to "**0**".
6. Status is one of:
    - **OK** – a valid symbol and market were specified and that client will start receiving asynchronous updates.
    - **ERROR|<reason>** - the symbol and/or market were invalid and the client will receive no update. **<reason>** indicates the reason for the error and is one of:
        1. **invalid symbol**
        2. **invalid market**

1.2.2.2 Full Quote Message

Full quote messages are sent whenever the full quote state of a symbol/market change.

These message have the form:

```
NT~B~Q~U~0~Symbol
```

The 6 fields of the message are:

1. Symbol
2. Market
3. "**Q**" to indicate that this is a Full Quote message.
4. Operation is not applicable and always set to "**U**".
5. Offset is the full quote value offset as described below, starting at 0(zero). Even values are Full Quote field names and the odd values are the Full Quote field values, where N (even) is the field name and N + 1 (odd) the the associated value.
6. Value is the Full Quote field name or value.

Offset	*Description*
0	Symbol field name.
1	Symbol field value.
2	Boardlot field name.
3	Boardlot field value.
4	Currency field name.
5	Currency field value.
6	CUSIPld name.
7	CUSIP field value.
...	TBD

Full quote values are only sent when the value change, so the field names (offset is odd) are only sent once, and the field values are only sent when they change. Some of the field values, for example symbol, currency etc. are only sent once as

they never change, others such as LastSale or LastSaleTime are send after each trade.

1.2.2.3 Bid/Ask Messages

Full quote messages are sent whenever the full quote state of a symbol/market change.

These message have the form:

```
NT~B~A~U~89~7|3.400|16000|
```

The 6 fields of the message are:

1. Symbol
2. Market
3. "**B**" to indicate that this is a Bid/Buy message or "**A**" to indicate that this is an Ask/Sell message.
4. Operation is "**U**" to update the price level specified by **Offset**, "**I**" to insert a new price level after **Offset**, or "**D**" to delete the price level at **Offset**.
    - **U** – the value at **Offset** is changed to the specified **Value**. If there is no current value for the price level at the specified **Offset**, it is created.
    - **I** – a new price level is inserted after **Offset** with the specified **Value**, all existing price level(s) after **Offset** are pushed down by one level.
    - **D** – the price level at **Offset** is deleted, all prices level(s) after **Offset** are pulled up by one.
5. Offset is the index into the Bid or Ask side of the MBP, starting at 0(zero).
6. Value is the new value associated with the Bid or Ask price level at the specified **Offset**. This field consists of 4 "|" (ASCII decimal 124, hex 7C) fields:
    - number of orders at this price level
    - price at this price level
    - volume at this price level
    - terms at thus price level

4.2 Reading Messages from the Market By Price Service

The client application reads the stream of MBP data from the TCP feed socket per the socket API being used, i.e. **read**() or **recv**() if the Berkeley Socket API is being used.

The client should always be ready to read data from the socket as it arrives, either using the **select**() function to multiplex IO on multiple file descriptors or using a dedicated thread in a multi-threaded program.

Should the client application fail to read data from the MBP socket in a timely manner, the Market By Price Managed Service may disconnected the socket on its end if the the socket connection is consuming too many resources.

Typically the client application will not receive one complete MBP message per call to **read**() or **recv**(). The client application must build complete newline delimited MBP messages from the MBP data stream.

Once a complete MBP message is received, it can be tokenized as described above.

4.3 Disconnecting from the Market By Price Service

The application closes the TCP socket connection to the Market By Price Service.

1.7 Microsoft Excel RTD Interface

1.7.1 Overview

The Micro-line Market By Price Service provides an Excel RTD interface to (TSX/TSXV) MBP (Boardlot, Oddlot and Terms) order books.

The RTD interface consists of a DLL file and registry (.reg) definition files that are copied via HTTP from the Micro-line Market By Price Service and installed on each client workstation that uses the RTD interface.

The Micro-line MBP RTD interface is started by Excel each time a spreadsheet containing Micro-line MBP RTD formulas is loaded.

1.7.2 Limitations

The Microsoft RTD Interface uses a push/pull model. The Micro-line RTD server informs Excel when there is an update available for a cell. Excel will ask the RTD server for updates when it is ready.

From the Microsoft RTD FAQ:

How Do I Make Sure That Excel sees Every Update?

It's basically up to the RTD server to decide what to do with the values if more than one update on a particular topic comes in before Excel calls back to the server for updates. Generally, we expect servers to just throw out the old value and pass the new value into Excel when it asks for updates. But there are some instances we've heard of where someone wants every update. In that case, the RTD server needs to queue up updates. When Excel asks for updates, the RTD server sends Excel the oldest one in the queue. The server continues calling the ***UpdateNotify*** *method while the queues still have values in them.*

The Micro-line RTD server throws away old values per Microsoft best practices for RTD. If it were to queue values for rapidly updating instruments, then the RTD server would present the client application with out of date data.

Any client application that needs guaranteed delivery of tick by tick market data should not be implemented in Excel but should use the **TCP Socket Interface** described above.

1.7.3 Hardware and Software Requirements

Microsoft Excel version 2002 (office XP) or above running under Microsoft Windows 98, ME, XP or 2000. The RTD service cannot be used on earlier versions of Excel as the real-time data technology required for a robust streaming service was first released in Excel 2002.

What Kind of Performance Statistics Have Been Seen With the New RTD Architecture?

On a Pentium III 500 MHz processor with 128 MB of RAM, 20,000 unique topics can be updated three times per second; one topic can be updated 200 times per second.

How Frequently Can Excel Accept Updates?

These are the numbers that we get with an RTD server running on the same computer that Excel is on.

Based on a Pentium III 500 MHz processor with 128 MB of RAM, 20,000 unique topics can be updated three times per second; one topic can be updated 200 times per second.

The number of times that a single topic can be updated seems to be limited by the number of times that Excel checks for Microsoft Windows messages, which is at most 700 times per second. Since some of the messages have higher priority than RTD does, Excel effectively gets about 200 updates per second.

1.7.4 Installation and Configuration

1.7.4.1 Location of Market By Price Service

1.2.2.4 Refer to the *"Administration – Market By Price Managed Service"* document for the location of the market By Price Service, referred to as "**<MLMBP_IP>**" below.

1.7.4.2 Create Local Directory/Upgrade

Create a directory on the client workstation called "**C:\MLMBP**".

If this directory already exists then this an upgrade. In this case, execute the following commands from a DOS shell to unload the existing RTD server after shutting down all Excel spreadsheets:

```
cd c:\MLMBP
regsvr32 /s /u MLRTDMBPServer.dll
```

1.7.4.3 Registry

The following registry files are copied from the Micro-line Market By Price Service:

- http://<MLMBP_IP>/rtd.mbp.reg – MBP RTD server registry settings. Copy to the "**C:\MLMBP**" local directory on the client workstation and double click from Windows Explorer. This file contains IP addresses and ports that allow the MBP RTD server to connect to the MBP Service.
- http://<MLMBP_IP>/excel.rtdthrottleinterval.reg – Sets the Excel RTD throttle interval to zero milli-seconds so that TSX CL1 and TL1 quote and trade tics are delivered to Excel with no delay. The default value is 2000 mill-seconds, which is of no use to an application that needs tick by tick data in real time. Copy to the "**C:\MLMBP**" local directory on the client workstation and double click from Windows Explorer.

1.7.4.4 DLL

Copy the file "**http://<MLMBP_IP>/MLRTDMBPServer.dll**" the "**C:\MLMBP**" local directory. Execute the following commands from a DOS shell:

```
cd c:\MLMBP
regsvr32 /s MLRTDMBPServer.dll
```

These commands must be executed each time the client workstation is rebooted.

1.7.4.5 Set Excel Security Level

The behaviour of the RTD server is governed by Excel security settings. To examine or modify your security settings in Excel, on the **Tools** menu, click **Macro**, and select **Security**.

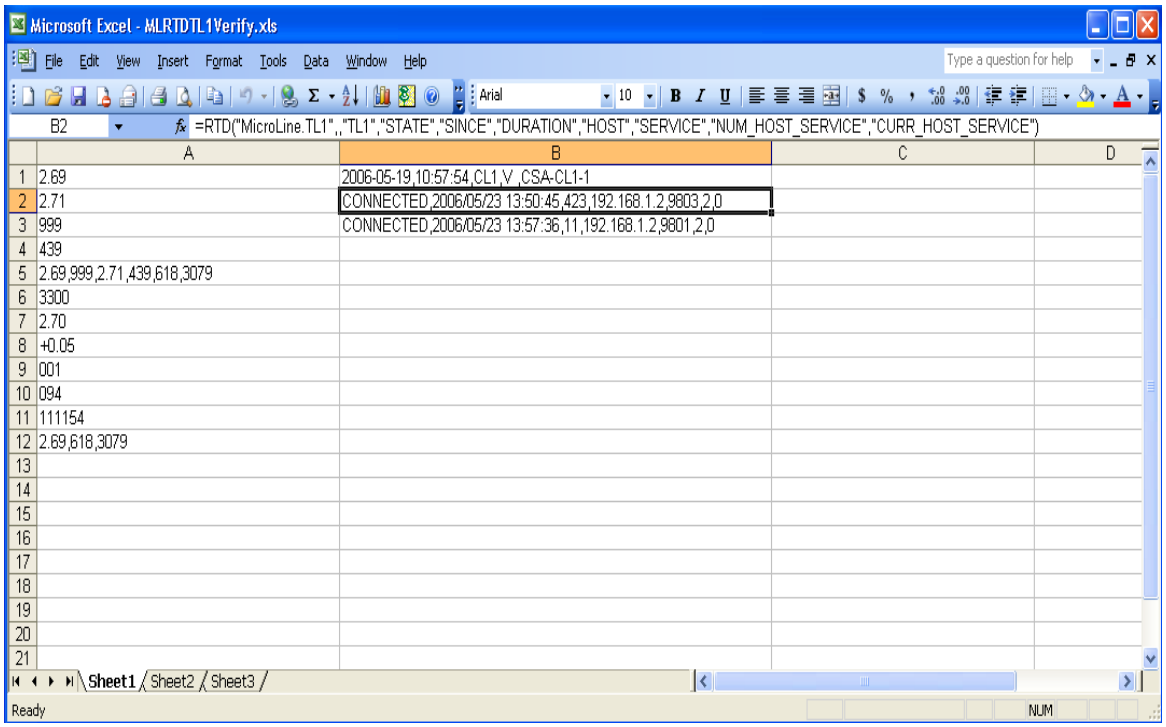
- When the security is set to **High**, the RTD server fails silently, and Excel displays #N/A in the worksheet cell.
- When the security is set to **Medium**, Excel 2002 displays a warning that the RTD server contains macros and asks the user whether to enable or disable them. If the macros are enabled, the RTD server runs normally. If the macros are disabled, the RTD server fails silently and Excel displays #N/A in the worksheet cell.
- When the security is set to **Low**, the RTD server runs normally without any security warnings.

Excel 2002 is designed to work in this manner for security purposes. Because an RTD server can be installed on a computer without the user's knowledge, and because an RTD server requires no user interaction to function, Excel 2002 notifies users when an RTD server is present and allows them to disable it if they wish.

1.7.4.6 Verify Installation and Troubleshooting

Copy the file "**http://<MLMBP_IP>/MLRTDMBPVerify.xls**" to the "**C:\MLMBP**" directory.

Load the file into Excel and compare to the diagram below:



Excel has connected to the Micro-line MBP RTD server and is receiving updates. If this is being run during market hours (Mon – Fri 07:00 – 16:00), expect the see the cells update frequently.

If all the cells contain the value "**#N/A**" then there is a problem. Verify the installation steps above.

The most likely causes of the problem are:

- The registry values have not been set. Use **"regedit32"** to verify that the following values are set:

  1. HKEY_CURRENT_USER\Software\Micro-line\RTD\MBP\mbp_host – one or more comma separated IP addresses for the MBP service.

  2. HKEY_CURRENT_USER\Software\Micro-line\RTD\TL1\mbp_service – one or more comma separated ports for the MBP service.

- Re-run the registry files per the **Registry** section above. Refer to the "*Administration – Market By Price Managed Service*" document for details on valid IP addresses and ports for the above registry entries.
- Verify that Excel security setting is "**Medium**" or "**Low**".

- The file **"MLRTDMBPServer.dll**" has not been loaded. Run the command "**regsvr32 MLRTDMBPServer.dll**". A dialog box with the message "**DllRegisterServer in MLRTDMBPServer.dll succeeded**". Any other message indicates that the DLL is not loaded. Verify installation of the DLL as described above.
- Check the RTD server log file. This file is created in "**C:\MLMBP**" each time the RTD server is started and has a name of the form "mbp**_rtd_server-YYYYMMDD-HHMMSS.log**", i.e. "mbp**_rtd_server-20060523-131547.log**". This file will only exist if the file "**MLRTDMBPServer.dll**" has been loaded. Look for error messages of the form:

```
2006/05/23 13:15:47;error;VCRTDServer()::ServerStart():
RegQueryValue(HKEY_CURRENT_USER, 'Software\Micro-line\RTD\MBP\mbp_host') failed
```

In the above example, the RTD server is unable to locate a registry key.

1.7.5 Accessing MBP Data Using Excel

The general form of an Excel formula that provides access to the CL1 and TL1 data feeds is:

```
=RTD("MicroLine.MBP",,<TICKER>,<MARKET>,<LEVEL>,<FIELD>,...)
```

For example:

```
=RTD("MicroLine.MBP",,"NT","BOARDLOT", "0","ORDERS","VOLUME","PRICE")
```

displays the most recent "**ORDERS**", "**VOLUME**" and "**PRICE**" the first price level (**0**) for the BOARDLOT market for the ticker symbol "**NT**".

The "**http://<MLMBP_IP>/MLRTDMBPVerify.xls**" Excel verification spreadsheet demonstrates most of the RTD syntax described below.

The RTD function takes the following arguments:

1. A string that represents the Program ID of the MBP RTD server installed on the local system. It is always "**MicroLine.MBP**" for the MBP service.
2. The second argument is always empty as the RTD server is running on the client workstation so the name of a remote computer is not used.
3. A valid CL1(TSXV) or TL1(TSX) symbol **TICKER** in TSX ticker format or predefined values that allow the status of the RTD server to be displayed as described below.
4. A valid **MARKET**, one of **BOARDLOT**, **ODDLOT** or **TERMS**.
5. A **LEVEL** indicating which price level. The first price level, i.e. Highest bid or lowest ask, is 0 (zero).
6. One or more fields associated with the ticker symbol as described below.

The RTD function returns the specified fields into a single cell with multiple comma separated values, one per field in the RTD function call. It is the responsibility of the Excel spreadsheet to parse this into multiple fields if needed or to request only one field per cell.

The **FIELD**s are:

- ORDERS – The number of orders at the price level specified by **LEVEL**.
- VOLUME – The total order volume (shares not boardlots) at the price level specified by **LEVEL**.
- PRICE – The price at the price level specified by **LEVEL**.
- TERMS – The special terms at the price level specified by **LEVEL**, only defined for **MARKET** value of **TERMS**.
- COUNT – The total number of updates received for the **TICKER** by the MBP service since the RTD server has been started by Excel.
- MISSED – The total number of updates for the ticker symbol received by the MBP service since the RTD server has been started by Excel but missed by Excel, due to a new update being received from the MBP service before Excel has requested the previous update.

The following special ticker symbols allow the state of the RTD server to be displayed. These are typically used for diagnostic purposes:

- MBP – the state of the connection to the MBP service. The following fields are associated with this ticker symbols:
    - STATE – One of "**CONNECTED**" or "**DISCONNECTED**", indicating whether the RTD server is currently connected to the MBP service..
    - SINCE – The date/time that the RTD server entered the above STATE for the MBP service. The format is "**YYYY/MM/DD HH:MM:SS".**
    - HOST – The hostname/IP address that the RTD server is currently connected to for the MBP service if STATE is "**CONNECTED**" or the hostname/IP address that the RTD server last tried to connect to if STATE is "**DISCONNECTED**".
    - SERVICE - The service/port that the RTD server is currently connected to for the MBP service if STATE is "**CONNECTED**" or

the service/port that the RTD server last tried to connect to if STATE is "**DISCONNECTED**".

- NUM_HOST_SERVICE – The number of host/service pairs that the RTD server is configured to make connection attempts to as per the RTD server registry settings described above.
- CURR_HOST_SERVICE – The index of the host/service pairs that the RTD server is currently connected to for the "**CL1**" or "**TL1**" feed if STATE is "**CONNECTED**" or the index of the host/service pairs that the RTD server last tried to connect to if STATE is "**DISCONNECTED**". The values range from 0 (zero) to (NUM_HOST_SERVICE – 1).
- DURATION – The time in seconds since the RTD server entered the above STATE for the MBP service.