Lightspeed Blackbox Developers Kit (BDK)

Sample Trading Application Design Document

Version 1.00
June 20, 2011

To obtain additional copies of this document, contact:

Lightspeed Financial, Inc.
148 Madison Avenue, 9th Floor
New York, NY 10016
646-393-4815

About this Document

This document describes the design of the Sample Trading Strategy. The Sample Trading Strategy is the implementation of a simple trading strategy. The source code for the Sample Trading Strategy is provided with the Lightspeed Blackbox Developers Kit (BDK). When the Sample Trading Strategy code is compiled and linked with the BDK Library, the result is a fully functional Blackbox Trading System. Although the Sample Trading Strategy is a fully functional Blackbox Trading System, it should not be traded. The Sample Trading Strategy is provided to help software developers learn how to develop automated trading systems using the BDK.

Who Should Read this Document

This document is intended for software developers who will be developing automated trading systems using Lightspeed's Blackbox Developers Kit (BDK).

Terminology

Throughout this document the following terms are used:

Term	Definition
BDK	Blackbox Developers Kit. A set of tools that allow Lightspeed customers to quickly develop Blackbox Trading Systems.
BDK Library	A compiled form of the BDK software components common to all Blackbox Trading Systems.
Source Code	The mechanism used to specify the actions to be performed by a computer. Source code is in human readable form and must be compiled before it can run on a computer
Console	A software application used to configure, monitor and control a Blackbox Trading System.
Remote Console	A console that is running on a computer that is physically located at a different location than the computer the Blackbox Trading System application is running on.
Lightspeed's Blackbox Trading Environment	A set of services provided by Lightspeed that allow customers to develop Blackbox Trading Systems. These services include: market data service, order entry service, risk management, short availability service, etc..
Blackbox Trading System	A software application developed for the purpose of trading for a profit with little human intervention. Throughout this document the term

	"Blackbox Trading System" is used to refer to an application that is created by compiling the customer's software and liking it with the BDK Library.
Trading Strategy Logic	A component of a Blackbox Trading System that will analyze market conditions, and when conditions are met, will send buy and sell orders to the market.
Position Management Logic	A component of a Blackbox Trading System that will determine when positions will be covered.
Alternative Trading System	A SEC approved, non-exchange trading venue. Refer to Rule 300 (a) of the SEC's Regulation ATS for the legal definition.
API	Application Program Interface. A set of routines, protocols and tools for building software applications.
Lightspeed Gateway	Lightspeed's Order Entry Server. Blackbox Trading Systems connect to the Lightspeed Gateway to send orders to the market and receive status about orders from the market.
Short Availability	Information describing whether a security is eligible for short sale, and the number of shares that can be sold short.
Function Call	Refers to the customer's software calling a function provided by the BDK Library.
Call-Back Function	Refers to the BDK library calling a function provided by the customer's software.
Customer's software	Refers to software developed by the customer that is compiled with the BDK Library to create a Blackbox Trading System.
Console operator or Operator	The person responsible for starting, stopping, configuring and monitoring the Blackbox Trading System.
Sample Trading Application	The Sample Trading Strategy is the implementation of a simple trading strategy. The source code for the Sample Trading Strategy is provided with Lightspeed's Blackbox Developers Kit (BDK). When the Sample Trading Strategy code is compiled and linked with the BDK Library, the result is a fully functional Blackbox Trading System. The Sample Trading Strategy is not meant to be traded. The Sample Trading Strategy is provided to help software developers learn how to develop automated trading systems using the BDK.

Revision History

The table below records the revision history of this document:

Revision	Date	Changes Made
1.00	June 20, 2011	Initial Document

Contents

1. Introduction

This document describes the design of the Sample Trading Strategy that is provided with Lightspeed's Blackbox Development Kit (BDK). The source code for the Sample Trading Strategy is provided and should be used in conjunction with this design document.

This document is organized as follows:

Section 1 is the introduction.

Section 2 discusses the intended use of the Sample Trading Strategy and the source code files that make up the Sample Trading Strategy.

Section 3 provides a description of the trading algorithm.

Section 4 describes the data structures used in the Sample Trading Strategy.

Section 5 discusses program flow and thread issues.

Section 6 discusses how order and position state is rebuilt when the Blackbox Trading System is restarted.

Section 7 describes how to access the depth of book.

Section 8 discusses console communication.

Section 9 discusses miscellaneous topics that the developer should be aware of.

2. Sample Trading Strategy Purpose

The Sample Trading Strategy is the implementation of a simple trading strategy. It is intended to help software developers learn how to develop automated trading systems using Lightspeed's Blackbox Developers Kit (BDK).

The following components are required to build a Blackbox Trading System using the Sample Trading Strategy code:

BDK Library	The BDK Library is available for Linux and Windows platforms. liblsskd.a is for Linux, and lssdk.lib for Windows.
customer_sample.c	C Source code file for the Sample Trading Strategy.
customer_sample.h	Header file for the Sample Trading Strategy.
sdk_proto.h	Header file that contains prototypes for all the BDK defined function calls.
platform.h	Header file that defines which platform to build (Linux or Windows)

When the Sample Trading Strategy code is compiled and linked with the BDK Library, the result is a fully functional Blackbox Trading System. To run the Sample Trading Strategy program, the following configuration files are required.

sdk_cfg.1	Configuration file used by the BDK Library. This file contains all the configuration parameters needed by the BDK Library. The "Configuration Guide" document available on Lightspeed's website, describes each configuration parameter.
symbol.list.1	This file contains the list of symbols the BDK library is aware of. The BDK Library requires a list of all symbols that the customer's software may trade. One use of this list is allow the BDK Library to register to receive data.
symbol.conf.1	This file contains configuration information required by the Sample Trading Strategy program. This file is not used by the BDK Library. It is only used by customer's software. The contents of this file and how it is used by the Sample Trading Strategy is described in the next section.

3. Trading Strategy Algorithm

3.1. Disclaimer

The Sample Trading Strategy is not intended to make money. It is provided to demonstrate how an automated trading system can be built using the BDK.

3.2. Algorithm

The trading algorithm (often called the trading strategy) is very simple. The trading strategy can be divided into two algorithms. One algorithm is used to open positions and the other is used to manage positions.

3.2.1. Opening a Position

If the stock is trading above the previous day's closing price, then the Sample Trading Strategy will attempt to buy at the inside bid price. If the stock is trading below the previous day's closing price, then the Sample Trading Strategy will attempt to sell short at the inside offer price.

3.2.2. Managing a Position

Once a position is acquired, an order is entered to cover the position at a profit as defined by the target profit specified in the Sample Trading Strategy's configuration file (symbol.conf.1). The position exposure is constantly monitored and when the loss exceeds the stop loss amount specified in the configuration file, then the position is covered at a loss.

3.2.3. Configuration Information

Configuration information is stored in the Sample Trading Strategy configuration file. The file is called symbol.conf.1. The configuration file is read when the Blackbox application is started. The configuration file contains a list of stock to be traded. The configuration file also contains the target profit amount, the stop loss amount, and the maximum position size allowed.

The following is an example of a configuration file that contains two stocks to be traded.

```
MSFT,500,.05,.75
AAPL,100,.20.2.00
```

This configuration file instructs the Sample Trading Strategy to trade MSFT for a .05 profit with a stop loss of .75 with a maximum position size of 500 shares. It also says to trade AAPL for a .20 profit with a stop loss of 2.00 with a maximum position size of 100 shares.

3.2.4. Other Algorithm Details

- The Sample Trading Strategy will not attempt to add to an existing position. For example, assume an order to buy 500 shares of MSFT is placed. Also assume that the Sample Trading Strategy is notified that 100 shares are filled. The Sample Trading Strategy will cancel the buy order and manage the 100 share position.
- Orders to open a position area placed as INET hidden orders.
- Orders to cover a position at the desired target profit are placed as INET hidden orders.
- Orders to stop out of a position at a loss are entered as RASH orders.
- The Sample Trading Strategy will not hold positions over night. All positions are covered at the time "Stop Time" specified in the BDK configuration file (sdk_cfg.1). A Stop Trading Time can be specified in the BDK configuration file. When this time is reached, the BDK Library will notify the customer's software via the stop_time_event() callback function. The Sample Trading Strategy will cover all positions when the stop_time_event() callback function is called. Refer to the BDK API Specification for a discussion of the "Start Time" and "Stop Time" notification mechanism.
- The Sample Trading Strategy will not launch more than 30 orders per second. This is an arbitrary number and does not reflect the Gateway Order Processing System's ability to process orders. The Gateway Order Processing System can process more than 30 orders per second.
- The Sample Trading Strategy only requires the best bid and offer price to make trading decisions. However, section 7 describes how to access depth of book information.
- A "global" trading state is used and pertains to all symbols. Possible global trading states are:
    - Trade – Automated trading is enabled.
    - Don't Trade –Automated trading is disabled.
    - Liquidate Only – Automated trading can be used to cover existing positions, but cannot be used to open new positions.
    - Bail Out – In this mode, the Sample Trading Strategy will cover all open positions and when complete will stop (disable) automated trading.
- An "individual symbol" trading state is used and pertains to a specific symbol. Possible individual symbol trading states are:
    - Trade – Automated trading is enabled for the symbol.
    - Don't Trade –Automated trading is disabled for the symbol.

  - Liquidate Only – Automated trading can be used to cover an existing position, but cannot be used to open a new position for the symbol.

- Both the global trading state and the individual symbol trading state are checked when making decision regarding when automate trading can be done. The following table will help clarify:

Global Trading State	MSFT Trading State	Automated Trading Behavior for MSFT
Trade	Trade	Automated trading of MSFT is allowed
Trade	Don't Trade	Automated trading of MSFT is not allowed
Trade	Liquidate Only	Automated trading of MSFT is only allowed to cover an existing position
Don't Trade	Trade	Automated trading of MSFT is not allowed
Don't Trade	Don't Trade	Automated trading of MSFT is not allowed
Don't Trade	Liquidate Only	Automated trading of MSFT is not allowed
Liquidate Only	Trade	Automated trading of MSFT is only allowed to cover an existing position
Liquidate Only	Don't Trade	Automated trading of MSFT is not allowed
Liquidate Only	Liquidate Only	Automated trading of MSFT is only allowed to cover an existing position
Bail Out	Trade	Automated trading of MSFT is only allowed to cover an existing position, and the Sample Trading Strategy will aggressively attempt to cover all open positions.
Bail Out	Don't Trade	Automated trading of MSFT is not allowed.
Bail Out	Liquidate Only	Automated trading of MSFT is only allowed to cover an existing position, and the Sample Trading Strategy will aggressively attempt to cover all open positions.

4. Data Structures

4.1. Symbol Data Structure

The Sample Trading Strategy organized data based on the stock symbol. A C structure is used to store all information for a symbol. All information related to a symbol is stored in the data structure below:

```
typedef struct _sym_t {
        struct _sym_t     *next_ptr;

        char              sec_id[SYM_SZ];
        long              sym_len;

        // config parms read from symbol.conf
        long              target_profit;
        long              stop_loss_amt;
        long              max_position_size;

        // miscellaneous
        char              security_trading_state;
        long              trading_letter_state;
        long              halt_flag;
        long              order_reject_trading_state;
        long              cancel_allowed_state;
        long              tot_shares_traded;
        char              short_avail;

        // position variables
        long              position_size;
        long              position_price;

        // order variables
        struct _order_t   *buy_order_ptr;
        struct _order_t   *sell_order_ptr;

        // P&L variables
        double            tot_bot_cost;
        double            tot_sold_cost;

        // market data
        long              best_bid;
        long              best_ask;
} sym_t;
```

The table below describes how each field in the symbol data structure is used.

Variable	Description
*next_ptr	Pointer used to create a linked list of symbol data structures.
sec_id	This variable is used to store the symbol name (eg, MSFT).
sym_len	The number of character in the symbol name. For example, this variable will contain 4 for MSFT.
target_profit	The target profit for the symbol. This value is read from the configuration file (symbol.conf.1) during initialization.
stop_loss_amt	The stop loss amount for this symbol. This value is read from the configuration file (symbol.conf.1) during initialization.
max_position_size	The maximum position size for this symbol. This value is read from the configuration file (symbol.conf.1) during initialization.
Security_trading_state	The individual symbol's trading state. Possible states are Trade, Don't Trade, and Liquidate Only.
trading_letter_state	Trading can be enable or disabled based on the first letter of the symbol. This field indicates is this symbol is enabled or disabled.
halt_flag	Indicates if trading is halted.
order_reject_trading_state	Used to indicate that an order was rejected. If an order is rejected, the Sample Trading Strategy will use this variable to reduce the number of orders entered for this symbol. There is no point in launching the same order over and over again having it rejected each time.
cancel_allowed_state	Used to indicate that a cancel request was rejected. If a cancel request is rejected, the Sample Trading Strategy will use this variable to reduce the number of cancel requests entered for this symbol. There is no point in launching the same cancel request over and over again having it rejected each time.
tot_shares_traded	The total number of shares traded.
short_avail	Indicates whether the symbol can be sold short. Possible value are: Y – short selling allowd X – short selling not allowed

	H – Hard to borrow T – Threshold N – Unknown
position_size	Indicates the current position size. A long position is represented with a positive value. A short position is represented with a negative value. A zero indicates no position.
position_price	Indicates the current position price. Contains the average position price.
*buy_order_ptr	This variable is a pointer to a list of data structures and each data structure represents one live buy order. The order data structure is described below.
*sell_order_ptr	This variable is a pointer to a list of data structures and each data structure represents one live sell order. The order data structure is described below.
tot_bot_cost	This variable contains the accumulated total bought cost. Each time a buy order is filled, the price is multiplied by the number of shares bought and the resulting value is added to this field.
tot_sold_cost	This variable contains the accumulated total sold cost. Each time a sell order or a short sell order is filled, the price is multiplied by the number of shares sold and the resulting value is added to this field.
best_bid	The best bid price.
best_ask	The best offer price.

4.2. Order Data Structure

The Sample Trading Strategy uses the following data structure to represent an order.

```
typedef struct _order_t {
        struct _order_t  *next_ptr;
        long             order_price;
        long             order_size;
        long             order_state;
        char             order_id[ORDER_ID_SZ];
} order_t;
```

Each time an order is launched, an order data structure is allocated to maintain information about the order. Each symbol data structure (described above) contains a pointer to a list of buy orders and a list of sell orders.

The table below describes how each field is the order data structure is used.

Variable	Description
*next_ptr	Pointer used to create a linked list of order data structures.
order_price	The order price.
order_size	The order size
order_state	The order state. The following order states are defined. NO_ORDER: Indicates that the data structure does not represent a live order. WAITING_ACCEPT: The order has been sent to the Gateway Order Processing System, and the Sample Trading Strategy is waiting for the order to be accepted. LIVE_ORDER: The order has been accepted by the Gateway Order Processing System. The order is now a live order. CANCEL_PENDING: The Sample Trading Strategy has sent a cancel request to the Gateway Order Processing System and is waiting to be informed that the order has been canceled.
order_id	A 5 character identifier chosen by the BDK to uniquely identify the order.

When the Sample Trading Strategy is started, it will allocate a fixed number of order nodes and put them on a list known as the free order list. This is done so that order nodes can be quickly allocated

with out having to call malloc(). When an order is launched, an order data structure is obtained from the free order list. The order data structure is filled in and put on either the symbol's buy or sell order list depending on the type of order. When an order is rejected, canceled or executed in full, the order data structure is removed from the symbol buy or sell list and returned to the free list.

4.3. Hash Table

A hash table is used to organize symbol data structures and to allow a symbol data structure to be quickly located given the symbol. A hash function is performed on the symbol and the result is an index into the symbol hash table array. The hash table is an array of pointers that point to symbol data structures. If multiple symbols hash to the same array index, then a linked list of symbol data structures is maintained. The pointer in the hash table array points to the first symbol data structure on the list.

The figure below illustrates the symbol hash table. It also illustrates the concept of multiple symbol data structures that hash to the same array index and form a linked list of symbol data structures. The figure also illustrates how buy and sell orders are kept on a linked list for each symbol.

In the figure below, MSFT, AAPL and BAC all hash to the same index in the hash table array. Because they all hash to the same array index, a linked list is created with three symbol data structures on the list. Note, in reality MSFT, AAPL and BAC do not all hash to the same index. The figure is meant to illustrate the concept, not the results of the actual hash function.

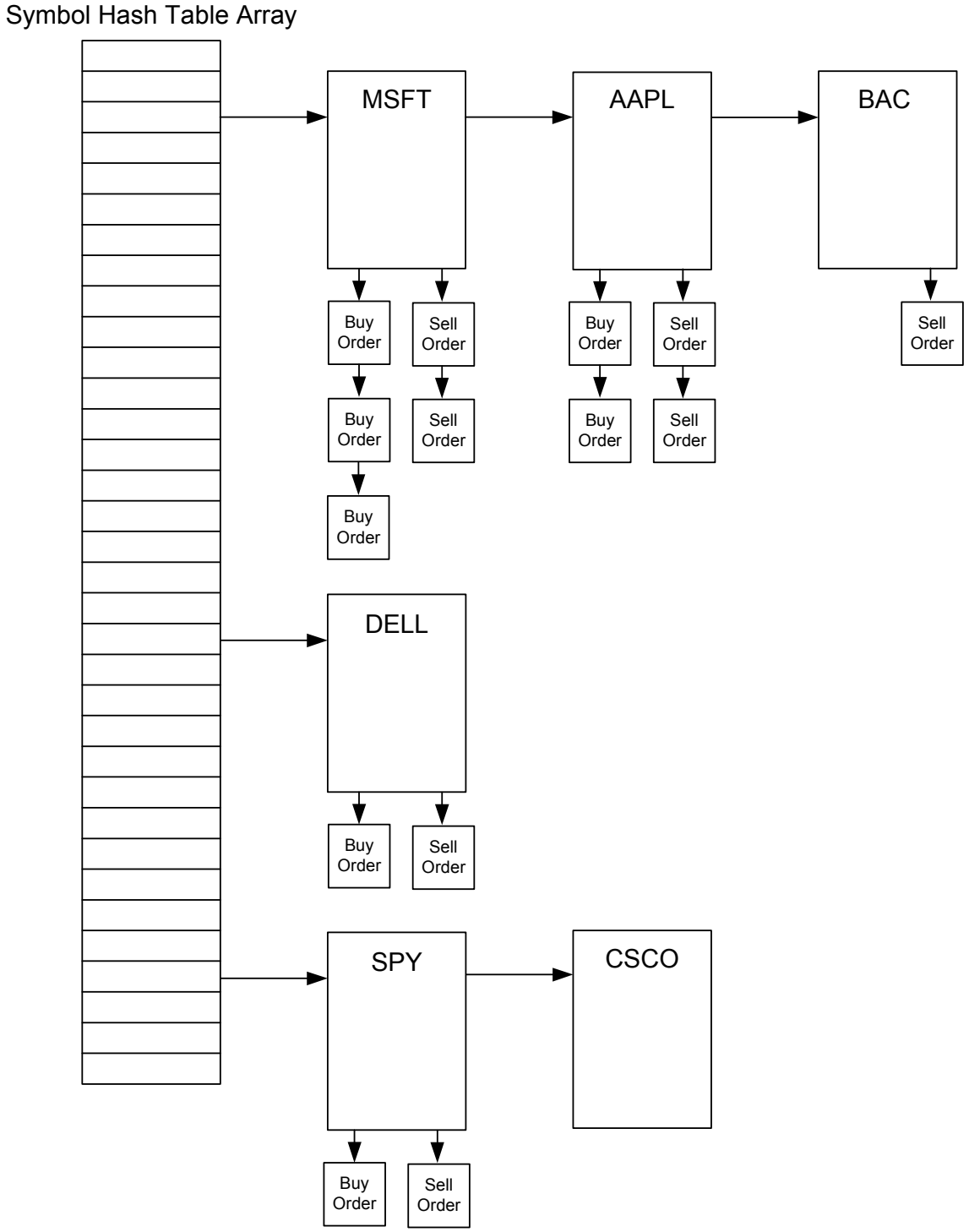
When the Sample Trading Strategy's initialize() callback function is called, it will read the configuration file (symbol.conf.1) and allocate a symbol data structure for each symbol in the configuration file. It will also insert the symbol data structure into the hash table as illustrated below. Refer to the load_config_file() functions in the Sample Trading Strategy code.

The figure below also illustrates the concept of "order lists". Each symbol data structure contains a pointer to a list of buy order data structures and a list of sell order data structures. In the figure below MSFT has 3 live buy orders and 2 live sell orders. BAC has only one sell order, and CSCO has no live buy or sell orders.

Order data structures are allocated when orders are launched. An order data structure represents a "working" order. When an order has been canceled or executed in full, it is removed from the order list.

The Sample Trading Strategy has a function called find_sym_node(). This function requires one parameter. This parameter is a char pointer that points to a buffer that contains the symbol. find_sym_node() will perform a hash function on the symbol and locate the symbol data structure. A pointer to the symbol data structure is returned. If the symbol data structure cannot be found, then NULL is returned. The following illustrates how find_sym_node() is used.

```
sec_ptr = find_sym_node(symbol);
```



5. Program Flow

5.1. Thread Model

Before discussing the program flow, it's important to understand how threads are used in the BDK Platform. When the Sample Trading Strategy source code is compiled and linked with the BDK Library, a single threaded application is created.

The application is single threaded and the main loop is implemented in the BDK Library. The Sample Trading Strategy code is invoked and can run when the BDK Library calls callback functions provided by the Sample Trading Strategy code.

There are seven types of events and will cause the BDK Library to call a callback function in the Sample Trading Strategy code. These seven event types are:

1) Initialization: When the Blackbox application is started the BDK Library will call the Sample Trading Strategy's initialize() callback function. This allows the Sample Trading Strategy to perform any required initialization.
2) Market Data Event: Examples of market data events that will cause the Sample Trading Strategy to be called include:
    a. A quote message is received that results in the customer's visible portion of the book having changed.
    b. A trade (print) is received.
    c. A trading halt message is received.
    d. A trading resume message is received.
3) Order Status Event: When the Sample Trading Strategy code launches an order, it will receive status about the order. For example, the Sample Trading Strategy will be called when an order is accepted, rejected, canceled, cancel rejected, or executed.
4) Timer Event: The Sample Trading Strategy can set a timer that will expire in the future. When a timer expires, the BDK Library will call the Sample Trading Strategy to notify it that the timer has expired.
5) Console Event: The Sample Trading Strategy will be notified via a callback function when the console operator has used the console to request that the Sample Trading Strategy take some action. Refer to the Console Communication section for a more detailed discussion regarding how the console and the Sample Trading Strategy interact.

6) Start of Day (SOD) Event: When the BDK Library connects to the Gateway Order Processing System, it will receive information regarding Start of Day positions and Start of Day Buying Power. The Sample Trading Strategy is notified of the SOD positions and buying power via several callback functions.

7) Restart Event: When the Blackbox Trading System is restarted, the BDK will process information in the .raw.x file and call the Sample Trading Strategy to notify it of SOD events and trading activity that occurred prior to the restart. When the Sample Trading Strategy is notified, it will rebuild its current order and position state. Refer to the Restart section for a detailed discussion regarding how to rebuild state when the Blackbox Trading System is restarted.

When a Sample Trading Strategy's callback function is called, it should perform its work as quickly as possible and return control to the BDK Library. Because the Blackbox Trading System is a single threaded application, control must be returned to the BDK Library as quickly as possible to allow the BDK Library to read data from its socket buffers. The BDK receives a large amount of market data from several market data feed connections and it must read data in a timely fashion.

5.1.1. BDK Library is Not Thread Safe

The BDK Library is not thread safe. If the customer's software creates a thread, then the thread should not call functions provided by the BDK Library. Doing so may cause undesired behavior and may cause the Blackbox Trading System to crash.

5.2. Program Flow

As mentioned above, the Sample Trading Strategy is called when one of the seven events described above occurs. There are two events that cause the Sample Trading Strategy to evaluate market conditions and make decisions regarding whether or not to have orders in the market. These two event types are Market Data Events and Order Status Events.

When the BDK Library receives a market data message that causes the customer's visible portion of the book to change, then the Sample Trading Strategy's quote_event_received() callback function is called. The quote_event_received() function will first locate the symbol data structure. Next it will save the inside bid and offer price in the symbol data structure. The best inside bid and offer prices are passed as parameters when quote_event_received() is called. Next the function modify_order() is called. The modify_order()function does the bulk of the work and makes decisions regarding whether or not to have orders in the market. The modify_order() function is described below.

When the BDK Library receives an order status message from the Gateway Order Processing System it will notify the Sample Trading Strategy by calling the appropriate callback function.

Examples of order status notifications include: an order was accepted, an order was rejected, an order was executed, an order was canceled, and a cancel request was rejected. When the Sample Trading Strategy is notified of an order status event, it will first locate the symbol data structure. Next the order is located and the order state is updated. Once the order state is updated, modify_order() is called. The modify_order() function makes decisions regarding whether or not to have orders in the market. The modify_order() function is described below.

5.2.1. modify_order() Function

The modify_order() function makes all decisions regarding when to have orders in the market. modify_order() will examine the market and the state of all orders and decide what action to take. For example, if there are no orders and the best bid price is greater than the previous day's closing price, then modify_order() will launch a buy order at the inside bid price.

Another example would be if there is no position, and one buy order with a state of LIVE_ORDER. The order price is now below the inside bid price. In this situation, modify_order() will launch a cancel request to cancel the order. Later when the Sample Trading Strategy is notified that the order is canceled, modify_order() will be called again. This time there will be no live orders and modify_order() will again compare the current price with the previous day's closing price and launch a new order.

For yet another example, assume the Sample Trading Strategy is notified that a buy order was executed in full via the order_executed() callback function. The order_executed() function will first locate the symbol data structure. It will then locate the order data structure for the order that has been executed. Since the order has been executed in full, the order data structure is removed from the symbol's buy order list. Once processing the order is complete, modify_order() is called. modify_order() will determine that there is a long position and no orders. modify_order() will launch a sell order with the price equal to the position price plus the desired target profit amount.

As you can see, modify_order() will examine the current position, orders, and market price and decide what action to take. When making decisions, modify_order() will also examine the conditions listed below:

- Global trading state
- Individual stock trading state
- Whether the stock is halted
- Whether automated trading has been disabled for all stocks based on the first letter of the symbol. For example, the console user can request that all stocks that start with the letter 'A' not be traded.
- Whether the "Trading Side" is enable or disables. For example, the console user can request that only long positions be acquired.

- Whether the allowed maximum number of orders per seconds has been reached. The Sample Trading Strategy only allows 30 orders per second to be launched.

The reader should refer to the source code to fully understand how modify_order() is implemented.

6. Restart Processing

When a Blackbox Trading System is started or restarted it must determine open positions and the state of all orders. This is sometimes referred to as rebuilding state for positions and orders.

When a Blackbox Trading System is started, the BDK Library will make a series of calls to callback functions provided by the Sample Trading Strategy code. The Sample Trading Strategy code can determine open positions and the state of all orders from the information provided in the callback functions.

How the Blackbox Trading System determines open positions and order state is best explained with an example.

In this example, the customer started their Blackbox Trading System in the morning and has two positions (DELL and MSFT) from the previous day. The customer enters a few orders and shuts down the Black Box Trading System before going to lunch. When the customer returns from lunch, they restart the Black Box Trading System.

Assume the following events occurred before the Black Box Trading System was restarted after lunch:

- Over night position in DELL, 100 share at 14.50
- Over night position in MSFT, 100 share at 25.00
- Order entered to sell 100 shares DELL at 15.00, good for the day
- Dell order accepted
- Order entered to sell 100 MSFT at 26.00, good for the day
- MSFT order accepted
- The DELL order to sell 100 share at 15.00 is filled
- Order entered to buy 200 shares of INTC at 12.00, good for the day
- INTC order accepted
- Cancel request entered to cancel INTC order to buy 200 shares at 12.00
- INTC order canceled
- The Black Box Trading System is shutdown

When the Black Box Trading System is restarted the Sample Trading Strategy callback functions will be called as follows:

- sod_positions(): The BDK Library calls the Sample Trading Strategy to inform it that there was an overnight position in DELL for 100 shares at 14.50
- sod_positions(): The BDK Library calls the Sample Trading Strategy to inform it that there was an overnight position in MSFT for 100 shares at 25.00
- sdk_restart_order(): The BDK Library calls the Sample Trading Strategy to inform it that an order was entered to sell 100 shares of DELL at 15.00
- order_accept():The BDK Library calls the Sample Trading Strategy to inform it that the order to sell 100 shares of DELL at 15.00 was accepted.

- sdk_restart_order(): The BDK Library calls the Sample Trading Strategy to inform it that an order was entered to sell 100 shares of MSFT at 26.00.
- order_accept(): The BDK Library calls the Sample Trading Strategy to inform it that the order to sell 100 shares of MSFT at 26.00 was accepted.
- order_execution(): The BDK Library calls the Sample Trading Strategy to inform it that the order to sell 100 shares of DELL at 15.00 was filled.
- sdk_restart_order(): The BDK Library calls the Sample Trading Strategy to inform it that an order was entered to buy 200 shares of INTC at 12.00.
- order_accept(): The BDK Library calls the Sample Trading Strategy to inform it that the order to buy 200 shares of INTC at 12.00 was accepted.
- order_cancelled(): The BDK Library calls the Sample Trading Strategy to inform it that the order to buy 200 shares of INTC at 12.00 was canceled.

The Sample Trading Strategy must rebuild its positions and order state from the information provided in the callback functions. The following positions and orders should be determined by the Sample Trading Strategy in this example:

- Open position in MSFT for 100 shares at 25.00
- Live order to sell 100 shares MSFT at 26.00, good for the day

The Sample Trading Strategy uses the same code to rebuild position and order state when the Black Box Trading System is restarted as it does when notified of order status events in real time. For example, the callback functions order_accept(), order_cancelled(), order_execution(), order_rejected(), and cancel_rejected() are called in real time and during restart processing.

The main difference is that during restart the callback function sdk_restart_order() is called to notify the Sample Trading Strategy that an order was launched earlier in the day. During normal trading, the Sample Trading Strategy simply calls launch_new_order() to launch an order. When sdk_restart_order() is called, the Sample Trading Strategy will allocate an order data structure, fill it in, and insert it on either the symbol's buy or sell order list.

7. Depth of Book

The Sample Trading Strategy makes its trading decisions solely on the best bid and offer price. It does not use depth of book information. However, many trading strategies require depth of book information, so it will be discussed here.

The BDK Library receives data from many sources and combines the data into an aggregated book. This is sometimes referred to as the consolidated book. The consolidated book contains one entry for each ECN and Exchange at each price level. An example of the consolidated book with 10 levels is shown below. Bids are on the left and offers are on the right.

```
500 INET  10.50       100 ARCA 10.55
200 NYSE  10.50       400 INET 10.56
100 BATS  10.50       300 BATS 10.56
800 ARCA  10.49       200 EDGX 10.56
500 CINN  10.49       100 ARCA 10.56
100 INET  10.49       100 INET 10.57
100 EDGX  10.49       100 NYSE 10.57
100 BOSX  10.49       200 INET 10.58
300 INET  10.48       100 ARCA 10.59
100 EDGA  10.48       100 INET 10.61
```

There is only one entry per price level for each ECN and Exchange. It's important to note that even though there are 10 "levels" in the consolidated book, there are only 3 prices levels for the bid and 6 price levels for the offer in this example. Within a price level, entries are sorted by size. In other words, the consolidated book is first sorted by price, and then by size.

The BDK Library provides a function called get_book_depth() to allow the customer's software to obtain access to the consolidated book To improve performance, the customer's software obtains pointers to the book maintained by the BDK Library. This method is more efficient than making a copy of the data each time the customer's software wants to examine the book. However, the customer's software must take care not to overwrite the data in the book.

The BDK Library maintains the consolidated book as an array of data structures of the following type. Two arrays are kept, one for bids and one for offers.

```
typedef struct  _data_book_t {
        char        mpid[4];
        long        i_price;
        long        size;
        long        ord_cnt;
} data_book_t;
```

To avoid doing pointer math, customers can access the consolidated book using array indexes. The following code sample demonstrates how this is done. This code sample is included in the Sample Trading Strategy code. It can be called to dump the contents of the consolidated book to the log file.

```
/*********************************************************************
| dump_book
|
| This routine will write the consolidated book to the log file.
*********************************************************************/
void
dump_book(char symbol[])
{
        long            depth;
        long            i;
        data_book_t   *bid_book;
        data_book_t   *ask_book;
        double          f_bid_price;
        double          f_ask_price;
        char            timeStr[20];
        long            sym_len;

        get_time_string(timeStr);

        depth = get_book_depth(symbol, &bid_book, &ask_book);

        if (depth > 4)
        {
                depth = 4;
        }

        sym_len = get_symbol_length(symbol);

        fprintf(fp_crit,"%s INFO: %*.*s  -------- consolidated book -------\n", timeStr,
                                        (int)sym_len, (int)sym_len, symbol);

        for(i=0; i<depth; i++)
        {
                f_bid_price = (double)bid_book[i].i_price / (double)ONE_DOLLAR;
                f_ask_price = (double)ask_book[i].i_price / (double)ONE_DOLLAR;

                fprintf(fp_crit, "%s INFO: %.2f/%ld/%4.4s/%ld     %.2f/%ld/%4.4s/%ld\n", timeStr,
                        f_bid_price, bid_book[i].size, bid_book[i].mpid, bid_book[i].ord_cnt,
                        f_ask_price, ask_book[i].size, ask_book[i].mpid, ask_book[i].ord_cnt);
        }
}
```

The dump_book() function demonstrates several important concepts.

1) get_book_depth() returns the depth of the consolidated book. The dump_book() function will only write 4 levels of the consolidated book to the log file.

2) get_book_depth() has three parameters:

    a. symbol is the string that contains the stock symbol

    b. bid_book is a pointer of type data_book_t. A pointer to this pointer is passed to get_book_depth(). get_book_depth() will return a pointer to the consolidated bid book.

    c. ask_book is a pointer of type data_book_t. A pointer to this pointer is passed to get_book_depth(). get_book_depth() will return a pointer to the consolidated ask book.

3) To avoid doing pointer math, customers can access the consolidated book using array indexes. For example, the following code is used to access the best bid price. Because the consolidated bid book is sorted, examining the 0^{th} element of the array will yield the best bid price.

    a. bid_book[0].i_price

4) Prices in the consolidated book are stored as integers with 4 digits of precision to the right of the decimal point. dump_book() converts the price to a floating point number before writing it to the log file. The following code converts the price to a floating point number.

```
#define ONE_DOLLAR 10000
```

    a. f_bid_price = (double)bid_book[i].i_price / (double)ONE_DOLLAR;

5) get_book_depth() does not need to be called each time the customer wants to access the consolidated book. The BDK Library always stores the consolidated book in the same memory location and the depth does not change. The customer could call get_book_depth() once and save the depth, bid book pointer and ask book pointer in the symbol data structure for later use. The reason for doing this would be to avoid the overhead of calling get_book_depth() each time the customer's software wants to access the consolidated book.

8. Console Communication

8.1. Console Communication Model

It's important to understand how the Console communicates with the Sample Trading Strategy code. The console is simply used to notify the Sample Trading Strategy code that the console user would like some action to be taken by the Blackbox Trading System. There are two methods by which the console user can notify the Sample Trading Strategy code that some action is desired.

8.1.1. Callback Functions

This is best described with an example. Assume the console user wants to notify the Sample Trading Strategy code that automated trading should only be used to cover existing positions and not to open new positions. This is often referred to as Liquidate Only Mode. In other words, the console user wants to notify the Sample Trading Strategy code to implement Liquidate Only Mode.

The process starts when the console user clicks the "Liquidate Only Now!" button on the console. The console will create a message that indicates the user has requested Liquidate Only Mode. The messages is sent to the Blackbox Trading System and received by the BDK Library. The BDK Library will examine the message and determine it is requesting Liquidate Only Mode. The BDK Library will then call the callback function update_global_trading_state() to notify the Sample Trading Strategy code.

The Sample Trading Strategy code should take steps to only cover existing positions and not open new positions (Liquidate Only). It is the Sample Trading Strategy code that makes trading decisions, not the BDK Library. So, only the Sample Trading Strategy code can implement Liquidate Only Mode. The BDK Library simply notifies the Sample Trading Strategy code. The Sample Trading Strategy code can chose to ignore the notification. Of course, this is not recommended.

> **IMPORTANT CONCEPT:** The BDK Library simply notifies the Sample Trading Strategy code based on messages it receives from the console. The Sample Trading Strategy code is responsible for acting on the notification and performing the desired behavior.

8.1.2. Console Messages

The second method used to notify the Sample Trading Strategy code to take action is to deliver console messages directly to the Sample Trading Strategy code. When the BDK Library receives a message from the console, it will examine the message. If the DBK Library does not recognize the

message type, it will call the callback function console_msg_received() and pass the message to the Sample Trading Strategy code.

The console command line interface allows the console user to send messages to Sample Trading Strategy code. The console application that is provided with the BDK contains a command line interface. The command line interface allows the user to type in "commands". The commands will be put into the message format described in the BDK API Specification and sent to Sample Trading Strategy code. The message type will be 5999. The Sample Trading Strategy code can assume that messages received with message type 5999 contain information entered using the command line interface. The code below is the console_msg_received() callback function provided with the Sample Trading Strategy.

```
/*********************************************************************
| console_msg_received
|
| This routine is called when a message from the console has been received.
*********************************************************************/
long
console_msg_received(char msg[], char **rtn_msg_ptr)
{
        long msg_type;

        msg_type = asctol(&msg[4], 4);

        switch (msg_type)
        {
        case 5999:
                // process the command line command
                if (memcmp(&msg[8], "dumpBBstats", 11) == 0)
                {
                        dump_all_stats();
                }
                break;

        default:
                log_data(fp_crit,"INFO: Unknown Console Command: Msg type: %d\n", msg_type);
                break;
        }

        return(0);
}
```

In the code sample above, only one console command is implemented (dumpBBstats). If the console message contains the string "dumpBBstats", then the function dump_all_stats() is called.

8.2. Console Display

The console simply displays information that is provided by the Blackbox Trading System. The information provided by the Blackbox Trading System can originate from either the BDK Library or the Sample Trading Strategy code.

Information such as P&L, exposure, positions, shares traded, fees, commissions, live orders, and market data status are provided by the BDK Library. Some of this information (P&L, exposure, shares trade, etc.) could have been provided the Sample Trading Strategy code, but is provided by the BDK Library to reduce the amount of coding required by the customer. Market data information can only be provided by the BDK Library, because the BDK Library provides all the code required to obtain market data.

Some of the information displayed on the console is provided by the Sample Trading Strategy code. The BDK Library will query the Sample Trading Strategy code for the information and forward the information to the console to be displayed. The following information is provided by the Sample Trading Strategy code:

- Sample Trading Strategy version number.
- Global trading state
- Individual stock (symbol) trading state
- The allowed trading side (long only, short only, either)
- Enabled and disabled trading letters. Trading of stocks can be enabled or disabled based on the first letter of the symbol. For example, don't trade stocks that start with the letter 'A'.
- Console down mode. Console down mode indicates whether the Sample Trading Strategy will allow automated trading when no consoles are connected to the Blackbox Trading System. Trading with no consoles connected to the Blackbox Trading System is referred to as "trading blind". The Sample Trading Strategy will not trade blind. Automated trading will be disabled if no consoles are connected.
- Trading Mode. Trading mode refers to whether the Sample Trading Strategy will start (enable) and stop (disable) automated trading automatically based on the start_time_event() notification and the stop_time_event() notification, or whether it requires the console operator to manual start and stop trading. The Sample Trading Strategy code requires the human console operator to manually start automated trading. The Sample Trading Strategy code disable automated trading and cover all positions when the stop_time_event() callback function is called.

  IMPORTANT CONCEPT: The information listed above is provided by the Sample Trading Strategy code. The BDK Library simply queries the Sample Trading Strategy code for the information and passes it to the console for display.

9. Miscellaneous Topics

9.1. Running Multiple Instances of the Blackbox Trading System

It is possible to run multiple instances of the Blackbox Trading System on one computer. As mentioned above, the BDK Library requires two configuration files: sdk_cfg.1 and symbol.list.1. Notice how both configuration file names ends with the character 1.

When the Blackbox Trading System is started it must be passed one parameter. This parameter is referred to as a command line parameter. For example, if the name of the Blackbox Trading System is bot, then it would be started as follows on a Linux platform.

```
./bot 1 &
```

The command line parameter of 1 instructs the Blackbox Trading System to use configuration files that have a 1 as the last character.

To run a second instance of the Blackbox Trading System would require a second set of configuration files. For example, the second instance could use the configuration files sdk_cfg.2 and symbol.list.2. The second instance of the Blackbox Trading System would be started as follows:

```
./bot 2 &
```

The command line parameter is used to specify which configuration files are to be used. There can be up to 10 instances of the Blackbox Trading System running on one computer. Valid command line parameters are 0 through 9.

NOTE: when considering to run multiple Blackbox Trading Systems on one computer, memory and cpu usage should be examined to ensure adequate resources are available to support multiple instances.

9.2. Order Type Status

The Gateway Order Processing System accepts nine types of orders. The nine order types are listed below

Value	Description
'I'	INET
'A'	ARCA
'T'	BATS
'Y'	NYSE, SuperDot, DirectPlus
'H'	EDGE A
'G'	EDGE X
'R'	RASH
'E'	AMEX
'O'	BOST
'J'	Jefferies
'P'	BATY

When the BDK Library connects to the Gateway Order Processing System it will obtain information from the Gateway regarding which order types the Gateway will accept orders for. The BDK Library will pass this information to the Sample Trading Strategy code by calling its order_type_status() callback function.

The console user can also change the status of an order type. For example, assume the console user wants to instruct the Sample Trading Strategy code to not launch ARCA orders. The console user can use the console application to cause a message to be sent to the Blackbox Trading System requesting that ARCA orders not be used. The BDK Library will process the message and call the Sample Trading Strategy code's order_type_status() callback function to notify it that ARCA orders should not be used.

> **IMPORTANT:** The Sample Trading Strategy code does not use the order type status information. The Sample Trading Strategy code launches two types of orders. INET order and RASH orders. The Sample Trading Strategy code assumes these order types are always enabled. This is not the most robust way to handle this situation. If the customer wants to make their trading strategy code more robust, they should check the order type status to ensure that the order type they want to use is enabled. If the desired order type is disabled, then they should chose an order type that is enabled.

9.3. Writing Information to the Console

A typical configuration is for the Blackbox Trading System to run on a computer that is co-located in Lightspeed's data center and the console is run on a computer at the customer's location. The

console and the Blackbox Trading System communicate using a tcp connection over the Internet. The Internet is slow compared to a Local Area Network (LAN). Because of the limited bandwidth of the internet, care must be taken when the Blackbox Trading System sends data to the console.

The log_data() function sends logging information to the console which is then displayed in the console scroll window. If care is not taken, it's possible for the Blackbox Trading System to send data to the console at a rate that exceeds the bandwidth of the Internet connection. If this occurs, the BDK Library's console socket buffer will fill up and the BDK Library will queue data at the application level. The BDK Library has a limit to the amount of data it will queue. If the queue limit is exceeded, then the BDK Library will drop the console connection.

To ensure that console connections are not dropped, customers should only send data to the console that is absolutely necessary. In other words, care should be taken when using the log_data() function

The log_data() function is used to have information displayed in the console scroll window, and optionally written to a file. In many cases is it better for the customer's Blackbox software to only write data to the log file, and not send it to the console as well. To only write to the log file, the log_data() function can be replaced with fprintf. The code sample below illustrates how to use fprintf to only write to the log file.

```
// write to log file and send message console
log_data(fp_crit, "INFO Hello there!\n");

// write to long file only
char  timeStr[20];
get_time_string(timeStr);
fprintf(fp_crit, "%s INFO Hello there!\n", timeStr);
```

9.4. Timer Usage

As mentioned above, a Blackbox Trading System built using the BDK platform is a single threaded application. Because the application is single threaded, the Sample Trading Strategy code must perform its work as quickly as possible and return control to the BDK Library. It is often desirable to break a large task into multiple smaller tasks. The BDK timer function can be used to do this.

How the Sample Trading Strategy code processes a console request to bail out of (immediately cover) all positions is an example of breaking a larger task into multiple smaller tasks. When the Sample Trading Strategy code is notified that it should cover all open positions, it could examine all symbol data structures, cancel all live orders and launch orders to cover all positions. When the task is complete it could return to the BDK Library.

Another approach is to examine only a small number of symbol data structures, cancel live orders and launch orders to cover positions. A timer is then set and when the timer expires, a few more

symbol data structures can be examined and processed. This process repeats until all positions have been covered.

The process_bail_out_timer() function in the Sample Trading Strategy code performs this algorithm. It will launch up to 5 orders to cover positions and up to 10 cancel requests. It will then set a timer for 2 milliseconds. When the timer expires two milliseconds later, it will be called again and repeat the process. This process repeats until process_bail_out_timer() is called and no positions exist. In this case the timer is not set because the task of covering all positions is complete.

9.5. Journal Position

The BDK platform supports the concept of journaling a positions. Journaling a position is typically used to correct an erroneous situation. For example, assume there was a clearing error and the Gateway Order Processing System reported a start of day position of 100 shares of DELL at 15.05 that was not a real position owned by the customer. To correct the situation, the console user could journal a position of -100 shares of DELL at 15.05 to offset the erroneous position.

When the console user journals a position, a message is sent from the console to the Black Box Trading System. The BDK Library will process the message and call the Sample Trading Strategy code's journal_position() callback function. The Sample Trading Strategy code will update the position state just as if it had been notified that an order was executed.

9.6. Start Of Day (SOD) Buying Power

When the BDK Library connects to the Gateway Order Processing System, it will receive the SOD Buying Power from the Gateway. The BDK Library will call the Sample Trading Strategy code's sod_buying_power() callback function to notify it of its SOD Buying Power.

The Sample Trading Strategy does not use the SOD Buying Power. The Sample Trading Strategy relies on the Gateway to reject orders if buying power is exceeded. However, customers may want to implement code to keep their remaining buying power and only launch orders if enough buying power exists to accommodate the order.

9.7. Market Data Status Events

When an event related to the Market Data system occurs, the BDK Library will notify the Sample Trading Strategy code of the event. Examples of events are a connection to a market data source has been made or dropped, the BDK Library is registering to receive market data, inactivity has

been detected on a connection, etc.. Refer to the BDK API Specification for a complete list of events.

The Sample Trading Strategy does not use market data status information provided in the mkt_data_status_event() callback function. However, to make their trading strategy code more robust, the customer may want to use this information. One idea would be to maintain state regarding the connections and registrations to all market data venues, and only allow automated trading if adequate data is available. For example, assume data is desired from INET, ARCA, BATS, EDGX, EDGA and the Quote Serer. Automated trading would be allowed if 4 or more of the data feeds are connected and registered. If less than 4 data feeds are connected and registered, then automated trading would be disabled.

9.8. Manual Orders Using the Console

The console user can manually enter orders and manually cancel orders using the console. When the console user wants to enter an order, a message is sent from the console to the Blackbox Trading System. The message contains all order parameters specified by the user. When the BDK Library receives the message, it will call the Sample Trading Strategy code's console_order_request() callback function and pass it all order parameters. The Sample Trading Strategy code will then call launch_new_order() to launch the order.

> **IMPORTANT CONCEPT:** The BDK Library does not launch the order. It simply notifies the Sample Trading Strategy code that the console user has requested that an order be sent to the market. The Sample Trading Strategy code must then launch the order.

Manual cancel requests are handled in a similar way. When the console user wants to cancel a single order or a group of orders, a message is sent from the console to the Blackbox Trading System. The message contains information indicating which order or orders should be canceled. When the BDK Library receives the message, it will call one of the Sample Trading Strategy code's callback functions that are used to cancel orders. The following callback functions are used to cancel orders:

console_cancel_all_orders()
console_cancel_symbol_all_orders()
console_cancel_one_order()

The Sample Trading Strategy code will then call launch_order_cancel() to launch the appropriate cancel requests.

9.9. Account Modes

The Gateway Order Processing System supports two modes of operation regarding clearing accounts. The two account modes are referred to as Single Account mode and Multi-Account mode. When the Gateway is configured for Single Account mode, it will put all trades into one clearing account. The customer does not need to specify an account number when sending orders to the Gateway. The Gateway will simply put all trades into the clearing account it was configured with.

When the Gateway is set up for Multi-Account mode, it is configured with a fixed number of clearing accounts. Orders received by the Gateway must specify the account the trade will be put into. When the Gateway receives an order, it will check the account number specified in the order. If the account number does not match one of the Gateway's configured account numbers, then the order is rejected.

The customer must specify whether the Blackbox Trading System will connect to a Gateway configured to operate in Single-Account mode or Multi-Account mode. This is done by setting a parameter (account-mode) in the configuration file (sdk_cfg.1). When Multi-Account mode is used, all account numbers that will be traded must also be specified in the configuration file. When the BDK Library connects to the Gateway, it will receive all account numbers the Gateway is configured with. The BDK Library will check to make sure the account numbers specified in the configuration file match the account numbers received from the Gateway. If there are account numbers in the configuration file that do not exist in the Gateway, then the BDK Library will log an error and exit. The Blackbox Trading System cannot be used to trade until the account numbers have been configured correctly.

The BDK Library's function calls and callback functions related to orders require an account number be passed as a parameter. If the Gateway and Blackbox Trading System are configured for Single Account mode, then calls by the customer's software to the BDK Library must set the account parameter set to 0 (zero). When the BDK Library calls the customer's software the account parameter will be set to 0. If the Gateway and Blackbox Trading System are configured for Multi-Account mode, then order related function calls and callback functions must pass a valid account number.

> **IMPORTANT:** The Sample Trading Strategy code has been written to support only Single Account mode. Single Account mode is much simpler when compared to Multi-Account Mode. If the customer is considering writing their trading strategy code to support Multi-Account Mode, they are encouraged to discuss their design with the BDK support team. The BDK support team can help the customer understand all the issues associated with developing trading strategy code that supports Multi-Account Mode.

9.10. Automated Trading with No Consoles Connected

The BDK platform supports a concept called console down mode. Console down mode indicates whether the Sample Trading Strategy code will allow automated trading when no consoles are connected to the Blackbox Trading System. Trading with no consoles connected to the Blackbox Trading System is sometimes referred to as "trading blind". The Sample Trading Strategy will NOT trade blind. Automated trading will be disabled if no consoles are connected.

The num_console_update() callback function is used to inform the Sample Trading Strategy of the number of consoles connected. The num_console_update() callback function is called each time a console connection is accepted or dropped. If the Sample Trading Strategy code is notified that there are no consoles connected, then it will disable automated trading.

9.11. Operational Procedures

Automated trading can be costly if something goes wrong. Before enabling automated trading, the user should ensure that the Blackbox Trading System is operating as desired. The following steps should be taken to ensure the Blackbox Trading System is functioning properly.

- Ensure all desired Market Data connections are up and registrations have been done. The user should view the book of several stocks using the “Trade Window” to ensure the market data looks correct.
- Ensure the connection to the Gateway Order Processing system is up. The user may want to enter a few orders off the market to ensure the orders are being processed correctly and that they can be canceled.